SISTEMELE DE OPERARE MIX

ŞI [redacted]

PROGRAMAREA ÎN LIMBAJUL MACRO

Pentru minicalculatoarele româneşti

Editura Tehnică

1

Ing. NICOLAE MANEA

MIX ŞI MACRO

VOLUMUL 1

SISTEMELE DE OPERARE MIX
pentru minicalculatoarele româneşti

EDITURA TEHNICA
Bucureşti

CUPRINS

CUPRINSUL VOLUMELOR 2 ŞI 3

VOLUMUL 2 : PROGRAMAREA ÎN LIMBAJUL MACRO

VOLUMUL 3 : TEHNICI DE PROGRAMARE ÎN LIMBAJUL MACRO

Prefața autorilor

O componentă de bază a programului economic și social din zilele noastre este UTILIZAREA PE O SCARĂ DIN CE ÎN CE MAI LARGĂ A MIJLOACELOR MODERNE DE PRELUCRARE AUTOMATĂ A DATELOR, în cele mai diferite domenii de activitate.

Din gama largă de tipuri de echipamente de tehnică de calcul astăzi fabricate de industria românească, de la microcalculatoare (industriale și personale) pînă la calculatoarele de capacitate medie-mare și o serie de echipamente periferice performante, FAMILIILE DE MINICALCULATOARE ROMÂNEȘTI (PE 16 BIȚI) de tip **INDEPENDENT** și **CORAL** ocupă un loc aparte prin performanțele lor tehnico-economice realizate și succesul lor în rîndul utilizatorilor, din țară și din străinătate.

Aceste minicalculatoare ocupă astăzi destul de detașat primul loc ca număr în cadrul parcului de calculatoare cu care este dotată economia noastră națională, interesul utilizatorilor pentru aceste tipuri de echipamente continuînd să se mențină la un nivel ridicat și chiar în creștere. Această adeziune și interesul utilizatorilor pentru minicalculatoarele românești **INDEPENDENT** *și* **CORAL** *se datorează atît calităților lor tehnice, proprii clasei de echipamente de calcul de tip minicalculatoare de generația 3, 5—4, cît și alinierii lor, atît din punct de vedere al compatibilității programelor cît și al arhitecturii și al tehnologiei utilizate, la standardele „de facto" impuse în acest domeniu pe plan mondial. Aceasta a permis trecerea în timp scurt la o producție industrială de serie a acestor echipamente, într-o varietate sporită de configurații și opțiuni, adaptate diferitelor tipuri de aplicații, precum și asimilarea rapidă a utilizării lor eficiente și programării acestor aplicații de către un număr tot mai mare de beneficiari. În economia noastră națională există actualmente* CÎTEVA MII DE MINICALCULATOARE DE FABRICAȚIE ROMÂNEASCĂ, *care se folosesc în cele mai diverse domenii de activitate, de la conducerea unor utilaje tehnologice (cu minicalculatoare în configurații adecvate, cu dispozitive periferice minimale) pînă la aplicații complexe de conducere și asistare a deciziei la nivelul unor întreprinderi, centrale industriale și instituții centrale, la nivelul economiei naționale (bazate pe minicalculatoare de mare productivitate, în configurații cu dispozitive periferice de mare capacitate, interconectate deseori în rețele complexe de minicalculatoare).*

La aceasta se adaugă şi cele circa o sută de minicalculatoare româneşti **INDEPENDENT** *şi* **CORAL** *instalate la beneficiari de peste hotare, livrate în mod frecvent împreună cu programele de bază şi aplicative corespunzătoare, sub forma unor* sisteme „la cheie", *orientate pe aplicaţii specifice, care valorifică superior atît realizările de prestigiu ale industriei româneşti de tehnică de calcul cît şi inteligenţa şi creativitatea specialiştilor noştri, înglobate în* programele aplicative *care determină în final utilitatea şi eficienţa utilizării echipamentelor de calcul. Această formă nouă de livrare a echipamentelor de calcul sub formă de sisteme „la cheie — orientate pe aplicaţii" concrete este o formă specifică minicalculatoarelor (precum şi microcalculatoarelor) fiind evident practicată şi pentru beneficiarii din ţară.*

Numărul mare de minicalculatoare şi mai ales diversitatea aplicaţiilor acestora, face ca aspectele legate de elaborarea programelor de bază şi mai ales a celor aplicative să dobîndească, o importanţă deosebită *atît sub aspect calitativ, cît şi cantitativ. Prin* aspectele calitative *ne referim la aspectele legate de domeniile de utilizare specifice, care necesită deseori performanţe deosebite pentru partea de programe, critice pentru aplicaţie (în general pentru aplicaţiile în timp real), diversitate de tipuri de funcţiuni realizate, calităţi deosebite privind interfaţa cu utilizatorii neinformaticieni, care, deseori, sînt în contact nemijlocit cu aceste echipamente de calcul (aplicaţii interactive, conversaţionale şi tranzacţionale, tipice pentru noile generaţii de minicalculatoare) ş.a.*

Prin aspectele cantitative *ne referim, pe de o parte, la numărul mare de aplicaţii, deseori asemănătoare între ele într-o anumită măsură, ceea ce impune o anumită tipologizare şi standardizare a soluţiilor şi modulelor de program, iar, pe de altă parte, raportul între costul total al unei aplicaţii şi costul asigurării programelor necesare acelei aplicaţii. Acest raport, în cazul minicalculatoarelor, datorită atît costului relativ mai redus al echipamentelor de tip minicalculator (faţă de minicalculatoarele de capacitate medie-mare la performanţe similare, acest raport este între 1 : 3 şi 1 : 5 în favoarea minicalculatoarelor) cît şi cerinţelor speciale privind performanţele programelor pentru multe din aplicaţiile minicalculatoarelor (cum sînt aplicaţiile „la cheie" sau cele în timp real), poate fi între 1 : 0,5 şi 1 : 0,8, cu tendinţă de creştere. Deseori, unele aplicaţii concrete ale minicalculatoarelor necesită o activitate amplă de* „inginerie a aplicaţiei", *care presupune atît o alegere a unor configuraţii adecvate de minicalculatoare (pe baza opţiunilor diverse pe care acestea le oferă) cît mai ales asigurarea programelor adecvate, prin generarea-adaptarea unor programe tipizate, inclusiv module funcţionale opţionale ale sistemului de operare, elaborarea de noi programe şi integrarea diverselor module de program într-un sistem unitar de programe, conform cerinţelor aplicaţiei.*

Programarea minicalculatoarelor, asigurarea programelor necesare unei anumite aplicaţii, utilizarea eficientă a multiplelor posibilităţi oferite de aceste echipamente de calcul cu flexibilitate ridicată, conferă o importanţă deosebită şi chiar noi valenţe activităţii de programare a aplicaţiilor bazate pe aceste tipuri de echipamente.

Lucrarea de faţă îşi propune să ofere unui cerc larg de cititori (utilizatori, programatori, studenţi) informaţiile generale şi specifice care să le permită atît o utilizare eficientă a minicalculatoarelor româneşti din familiile **INDEPENDENT** *şi* **CORAL**, *cît şi elaborarea de programe aplicative care să exploateze cît mai bine multiplele facilităţi evoluate ale acestor echipamente moderne de calcul.*

Cercul cititorilor interesați îl considerăm destul de larg, de cîteva zeci de mii, dacă socotim că în jurul fiecărui minicalculator deja livrat în economie sînt interesați în utilizarea și/sau programarea acestuia între 5 și 20 de persoane. La aceștia se adaugă cele cîteva mii de studenți care, anual, sînt învățați să lucreze și să programeze minicalculatoarele românești sau își elaborează lucrări de diplomă sau de cercetare, în cadrul institutelor de învățămînt superior (institute politehnice, universități, ASE) și chiar a unor licee industriale (licee de matematică-fizică cu profil de informatică). Să nu uităm totodată că multe din problemele prezentate în această lucrare își păstrează valabilitatea conceptuală și pentru alte modele și tipuri de minicalculatoare românești de mare performanță, pe 32 de biți, asimilate în fabricație, care au în mod justificat, unele elemente comune cu minicalculatoare românești actuale, pe 16 biți, mai ales din punctul de vedere al programării (inclusiv a limbajului de programare **MACRO** *și a altor componente conexe).*

Considerăm că această lucrare este așteptată de mai mulți ani de către cititori, interes constatat atît cu ocazia diverselor întîlniri între cei implicați (de exemplu, în cadrul Cercului Utilizatorilor de Minicalculatoare, a diverselor manifestări științifice organizate de către ITC, ICI, *și de către alte unități de informatică) cît mai ales din contactele directe cu utilizatorii care „asaltează" în continuare unitățile de specialitate* ITC *ICE) cu solicitări de documentație pentru utilizarea și programarea minicalculatoarelor.*

Această întîrziere în apariția prezentei lucrări este explicabilă parțial prin dificultatea de a elabora o asemenea lucrare care să reușească să concentreze și să organizeze un volum deosebit de mare de informații (amintim că manualele de referință și utilizare a sistemului de operare românesc **MIX/MIX-PLUS**, *de exemplu, reprezintă 34 de volume însumînd multe mii de pagini, la care se adaugă alte 49 de manuale referitoare la gestiunea datelor, comunicații și rețele, limbaje de programare de nivel înalt care însumează alte mii de pagini (vezi lista din anexa C la prezenta lucrare), precum și necesitatea unei suficiente acumulări de experiență practică în utilizarea și programarea acestor minicalculatoare și efortul foarte dificil de sintetizare a acestei experiențe dobîndite în diverse colective.*

Lucrarea de față își propune să ofere cititorilor informațiile necesare pentru a putea utiliza și elabora programe pentru minicalculatoarele românești în limbajul specific al acestora, denumit **MACRO**, un limbaj de programare deosebit de puternic cu unele facilități evoluate, moderne, care permite elaborarea unor programe care să realizeze maximum de performanțe posibile, asigurînd posibilitatea utilizării tuturor facilităților oferite atît de echipamentul fizic al acestor minicalculatoare cît și de sistemul lor de operare **MIX**.

O aplicație pentru minicalculatoare, în funcție de specificul aplicației și de pregătirea utilizatorului, poate fi programată folosind evident un limbaj de programare de nivel înalt (FORTRAN, COBOL, ADA, PASCAL, C, LISP etc.) sau limbajul de asamblare specific **MACRO** *(dacă se cer performanțe anumite*

în ceea ce privește memoria ocupată sau viteza de execuție) sau combinat, anumite părți (module) de program în limbajul de nivel înalt, iar altele în limbaj **MACRO**, *ceea ce constituie o altă calitate a sistemului de programare pe minicalculatoare. Dacă despre majoritatea limbajelor de programare de nivel înalt (FORTRAN, COBOL, PASCAL, LISP, ADA) există o anumită literatură minimală în limba română, despre limbajul de programare* **MACRO** *pentru minicalculatoare nu există publicată nici o lucrare în limba română, cu excepția unor suporturi de curs elaborate la ITC, ASE, IPB etc. și a manualelor de referință amintite mai sus, lucrări de care s-a ținut cont în elaborarea volumelor de față.*

O întrebare legitimă este dacă lucrarea de față poate ține locul celor cîteva mii de pagini cît cuprind manualele **MIX** *cu care se livrează minicalculatoarele. Lucrarea de față nu reprezintă, așa cum poate apare la prima vedere, o sinteză a manualelor de referință sau de firmă. Spre deosebire de manualele de referință amintite*, prezenta lucrare tratează și comentează în special aspectele conceptuale, arhitecturale și funcționale ale minicalculatoarelor românești în general și ale sistemului de operare (vol. I al prezentei lucrări) precum și conceptele, mecanismele, metodele și mijloacele de elaborare a programelor pentru aceste minicalculatoare, cu accent deosebit pe exemplificarea acestor posibilități prin programe concrete comentate.

Prin aceasta, lucrarea de față permite cititorului să înțeleagă conceptele de bază hardware și software ale minicalculatoarelor românești, să folosească în mod corect și eficient multiplele facilități de utilizare și programare oferite și să facă apel la manualele de referință mult mai rar, sau deloc. Desigur că manualele de referință sînt necesare și utile pentru a găsi o serie de amănunte concrete cînd se folosesc unele facilități speciale sau cînd apar unele erori în programe sau în funcționare mai subtile, ce trebuie lămurite. Informațiile dintre aceste manuale de referință sînt foarte amănunțite dar, de regulă, sînt prezentate direct, fără prea multe explicații sau comentarii, astfel încît utilizarea lor presupune cunoștințe generale deja însușite, scop urmărit de prezenta lucrare.

Volumul I *al lucrării se referă la* familia de sisteme de operare **MIX** ale minicalculatoarelor românești, *ale cărei concepte și posibilități trebuie bine cunoscute de orice utilizator sau programator de minicalculator.*

Într-un prim capitol se prezintă elementele arhitecturale ale minicalculatoarelor românești (magistrale de comunicații, organizarea memoriei și a accesului la memorie, conceptul funcționării stivei etc.) urmat de un capitol în care se prezintă conceptele și funcțiile sistemelor de operare **MIX** *pentru minicalculatoarele românești (compatibil și cu performanțe la nivelul unui sistem de operare reprezentativ, foarte răspîndit pe plan mondial, denumit RSX-1M al firmei DEC—SUA). Sistemele de operare* **MIX** *acoperă practic întreaga gamă de aplicații uzuale pentru minicalculatoare. Se prezintă principiile de multiprogramare și multitasking, planificarea task-urilor, mecanismelor de sincronizare și comunicare între task-uri, tipurile de evenimente, modalitățile de gestionare a memoriei și utilizare a spațiilor de adresare, particularitățile în exploatare a întreruperilor software, precum și alte concepte necesare a fi cunoscute în proiectarea și programarea unor aplicații.*

Accesul la resursele de calcul ale minicalculatoarelor este permis prin intermediul unor servicii specifice și interfețe generale sau particulare, ale sistemelor de operare **MIX**, *accesibile programatorului aplicației și care sînt prezentate în lucrare, în mod sistematizat.*

Un loc important în cadrul programării aplicațiilor utilizator îl ocupă sistemul de Intrare/Ieșire (I/E), datorită diversității dispozitivelor de I/E conectabile la minicalculatoare cît și flexibilității oferite de sistemul de operare în programarea intrărilor/ieșirilor. „Driverele" de I/E reprezintă interfețele sistemului de operare cu dispozitivele de I/E fizice. Lucrarea prezintă atît caracteristicile generale ale acestor interfețe, cît și facilitățile existente privind scrierea și depanarea unor drivere-utilizator.

O clasă importantă de servicii, puse la dispoziție de sistemul de operare, se referă la gestiunea și manipularea datelor. În lucrare se descriu serviciile standard puse la dispoziția utilizatorilor de gestiunea fișierelor **(FCS)** *și de gestiunea înregistrărilor* **(RMS).** *Lucrarea prezintă caracteristicile celor două tipuri de gestiune a datelor, organizarea fișierelor, modurile de acces la înregistrările fișierelor, formatul înregistrărilor, operațiile de I/E, interfețele de apel etc.*

În lucrare se prezintă, de asemenea posibilitățile de acces distribuit ale sistemului de operare **MIX** *în rețele de minicalculatoare. Se descriu succint : arhitectura rețelei de minicalculatoare* **MININET/MIX,** *conceptele de bază utilizate în configurarea și accesarea acestei rețele, precum și facilitățile generale : comunicarea între task-uri, accesul fișierelor de la distanță, transferul fișierelor la distanță, partajarea dispozitivelor de I/E la distanță, terminale virtuale, controlul task-urilor la distanță, comunicarea între terminale, accesul direct la liniile de comunicație, gestiunea și administrarea rețelei la distanță, încărcarea/evacuarea task-urilor la distanță, încărcarea sistemelor de la distanță.*

În ultimul capitol al volumului I se prezintă lista setului complet de manuale de referință și utilizare care însoțesc livrarea minicalculatoarelor românești.

Volumele II și III *ale lucrării cuprind în principal* descrierea limbajului de programare **MACRO** specific minicalculatoarelor românești *și mai ales* metode, tehnici și foarte multe exemple reprezentative de programare.

Fiind adresat direct programatorilor care doresc să realizeze aplicații performante, de mare eficiență, pe baza minicalculatoarelor, folosind în principal limbajul de programare **MACRO** *— care este capabil să folosească cel mai bine toate posibilitățile oferite de echipamentele fizice și sistemul de operare al minicalculatoarelor — volumul II debutează cu o prezentare a ceea ce reprezintă la ora actuală minicalculatoarele pentru utilizatori, structurile specifice și performanțele la care au ajuns, domeniile consacrate de utilizare, tendințele care se manifestă, urmată de o prezentare a principalelor caracteristici și tipuri de facilități oferite utilizatorilor de sistemele de operare moderne ale minicalculatoarelor actuale. Aceste capitole introductive completează cu informații la zi ceea ce s-a mai publicat în acest domeniu în literatura de specialitate din țara noastră și au menirea de a oferi cititorilor o viziune despre ceea ce le pot oferi astăzi minicalculatoarele și cum se plasează minicalculatoarele românești (descrise în vol. I, primul capitol) în raport cu nivelul atins pe plan mondial, atît în prezent cît și în perspectivă.*

Se prezintă în continuare caracteristicile și facilitățile limbajului de programare **MACRO**, *formatele diferitelor clase de instrucțiuni precum și utilizarea în programare a serviciilor sistemului de operare. Accent deosebit se pune pe metodele și tehnicile folosite în programare : tehnici de lucru cu macroinstrucțiuni definite de utilizator, realizarea structurată a programelor în limbajul* **MACRO**, *tehnici de lucru cu subprograme, elaborarea interactivă de programe în limbajul* **MACRO**, *scrierea de programe re-entrante în* **MACRO**, *utilizarea corutinelor, recursivitate, cod independent de poziție ș.a.*

Utilizarea tehnicilor de segmentare și a regiunilor statice și dinamice pentru date și programe — după cum se arată și se exemplifică în lucrare — oferă programatorului în limbajul **MACRO** *o gamă largă de servicii pentru realizarea unor aplicații eficiente.*

În continuare, sînt prezentate posibilitățile și modalitățile de utilizare în programarea în limbaj **MACRO** *a sistemelor de gestiune a fișierelor și respectiv a înregistrărilor, utilizarea rutinelor din biblioteca sistemului, interfața programelor scrise în* **MACRO** *cu programe scrise în alte limbaje de programare disponibile pe minicalculatoarele românești (FORTRAN, COBOL, PASCAL, C), facilitatea specială de preluare și prelucrare a liniilor de comandă.*

Volumul III conține cîteva capitole referitoare la scrierea de programe privilegiate în limbajul **MACRO**, *cum sînt procesoarele auxiliare, programe pentru comanda unor dispozitive periferice nestandard și pentru extinderea gamei de servicii oferite standard de sistemul de operare, fiecare problemă cu exemple concludente.*

Pentru programarea de aplicații în contextul utilizării minicalculatoarelor interconectate în rețele, în lucrare se descrie cadrul corespunzător de programare în limbajul **MACRO**, *cu accent pe comunicația între task-uri plasate în noduri diferite ale rețelei.*

Lucrarea prezintă într-un ultim capitol metode și tehnici utilizate în realizarea de aplicații relativ complexe, în timp real, pe baza experienței acumulate de către autori în elaborarea de sisteme de acest tip. O cerință frecventă este realizarea de aplicații specializate bazate pe sisteme de operare rezidente exclusiv în memoria internă (fără suport disc magnetic, deci fiabilitate mărită mai ales în condiții industriale de lucru) și în acest sens se prezintă cadrul de realizare și implementare a programelor în limbajul **MACRO** *dat fiind și particularitatea că în acest caz, dezvoltarea de programe se face pe un sistem „gazdă" cu disc magnetic, iar exploatarea aplicației pe un sistem „obiect" rezident memorie internă.*

În mod inerent, unele probleme prezentate în volumul I, sub formă de concepte și posibilități oferite, sînt reluate în volumele II și III din punctul de vedere al utilizării acestora în programarea în limbajul **MACRO** *și exemplificării prin programe concludente, comentate, toate implementate pe minicalculator și listate, deseori sub forma unor adevărate* studii de caz, *extrase din cazuri concrete și mai puțin scolastice, folosite în practica de cca. 10 ani a autorilor în proiecte complexe, implementate cu succes.*

Acest obiectiv, de ilustrare prin exemple concrete, concludente, izvorîte din practica concretă a unor aplicații complexe, urmărit în mod deliberat la elaborarea volumelor II și III a obligat antrenarea unui număr relativ mare de au-

tori (17 specialiști din cercetare-proiectare și învățămînt superior) unii dintre ei cu o experiență aproape unică sub anumite aspecte, ceea ce considerăm că reprezintă o calitate deosebită a lucrării și o detașează de orice altă lucrare scrisă pe acest subiect în literatura de specialitate pe plan mondial. Față de acest obiectiv ne-am asumat riscul dificultăților mari care apar în elaborarea unui volum cu peste 20 de autori și, cu tot efortul de coordonare și colaborare între autori, nu s-a reușit probabil o uniformizare perfectă a structurii capitolelor, a stilurilor de exprimare și uneori chiar a terminologiei, și poate chiar strecurarea unor mici imperfecțiuni în tehnoredactarea unor exemple de program (care, au fost de regulă testate cel puțin prin compilare pe minicalculator), neajunsuri pentru care ne cerem anticipat scuze cititorilor.

Un merit deosebit în realizarea prezentei lucrări revine conducerii ICI, ITC și ASE — Catedra de Cibernetică Economică care au susținut și sprijinit elaborarea acestei lucrări, precum și conducerii și redactorului de specialitate al Editurii Tehnice în care autorii au avut un sprijin deosebit.

1

Arhitectura minicalculatoarelor românești

1.1. Introducere

Deși inițial dimensionată pentru a satisface cu prioritate necesitățile de dotare a economiei naționale cu mijloace moderne de prelucrare a datelor, industria de tehnică de calcul din România a cunoscut o dezvoltare rapidă în special odată cu apariția mini și microcalculatoarelor (1976—1980).

Ca urmare a programelor adoptate în acest domeniu s-a trecut ulterior la producția de mare serie de minisisteme și echipamente de calcul pentru aplicații diverse, în special industriale, cu o pondere însemnată la export.

Un loc important în dezvoltarea tehnicii de calcul românești îl ocupă elaborarea unor *familii de minicalculatoare*, **INDEPENDENT** și **CORAL**, compatibile cu minicalculatoarele cele mai răspîndite și intens utilizate pe plan mondial.

La baza arhitecturii minicalculatoarelor construite în România, după o concepție originală, stă familia de minicalculatoare **PDP-11** a firmei Digital Equipment Corporation (**DEC**), SUA.

În același timp s-a ținut seama de compatibilitatea acestora cu cerințele formulate în cadrul Programului țărilor vecine de construire în comun a unor familii de minicalculatoare (**SM-3/SM-4/SM-5**) și sisteme de operare — **SMC**.

Corectitudinea acestei alegeri a fost și este confirmată atît prin larga utilizare a acestor echipamente în economia națională cît și prin exportul de minicalculatoare românești.

În domeniul cooperării în producție cu alte țări, România are o prezență activă, concretizată prin omologarea internațională a minicalculatoarelor **I-100** (1978), **I-102F** (1981), **I-106** (1986) și **DP15** (1989), precum și prin medalierea cu aur a minicalculatorului **I-100** (iunie 1979 — Moscova).

Au fost de asemenea omologate internațional o serie de echipamente de introducere de date, dispozitive periferice, microcalculatoare, memorii interne, sisteme de operare (**AMS, MINOS, MININET/MINOS, MIX, MIX-PLUS**) și programe-pachete de aplicații asociate.

1.2. Familia de minicalculatoare INDEPENDENT

Proiectată de către Institutul de Cercetări pentru Tehnică de Calcul **ITC**—București și fabricată de Întreprinderea de Calculatoare Electronice **ICE**—București și **IEPER**—București (**I-1016**), familia de minicalculatoare **INDEPENDENT** include patru tipuri de procesoare de bază, un mare număr de dispozitive periferice și suport software adecvat, reprezentînd un incontestabil succes și un produs reprezentativ al cercetării românești originale în domeniul tehnicii de calcul.

Procesoarele de bază din familia **INDEPENDENT** sînt următoarele :

— **I-100** ;
— **I-102F**, cu dezvoltările sale :
  — **I-102F** cu memorie extinsă ;
  — **I-102/4M** ;
— **I-106** ;
— **I-1016.**

Arhitectura comună pe care se bazează familia **INDEPENDENT** este impusă de proiectare și se reflectă în opțiunile de dispozitive periferice și în software.

În ultimul deceniu a avut loc o perfecționare continuă a caracteristicilor și facilităților sistemelor noi dezvoltate, pe baza progresului tehnologic înregistrat de industria producătoare de circuite integrate.

Aceasta a permis, în condițiile unei creșteri semnificative de performanțe, scăderea numărului plachetelor de circuite imprimate ale procesorului de bază, după cum urmează :

— **I-100 :** 13 plachete ;
— **I-102F :** 7 plachete ;
— **I-106, I-1016 :** 2 plachete.

Fundul de sertar de la **I-100**, realizat cu conexiuni cu fire („wire-wrapping"), a fost înlocuit la **I-102F, I-102/4M, I-106** și **I-1016** cu fund de sertar cu conexiuni imprimate, mai fiabil și mai ieftin, iar sursele de alimentare de la **I-100**, voluminoase (cu transformator), au fost înlocuite cu surse moderne, în comutație, de volum redus.

Odată cu miniaturizarea, inovațiile arhitecturale au permis includerea unor facilități noi, cum ar fi :

— microprogramarea dinamică, la **I-102F** ;
— diagnosticarea cu ajutorul unui procesor specializat, la **I-106** și **I-1016.**

Se poate concluziona de aceea, că dezvoltările avute în vedere în realizarea familiei de minicalculatoare **INDEPENDENT** au pus accent pe : compatibilitate, miniaturizare, funcționalitate, fiabilitate și testabilitate.

1.2.1. Minicalculatorul I-100

Tehnologic, minicalculatorul **I-100** utilizează circuite electronice TTL, precum și circuite MSI și LSI. Plachetele sînt de dimensiuni medii, utilizîndu-se tehnologia cu patru straturi ; fundul de sertar este cablat.

Minicalculatorul **I-100** fiind microprogramat dinamic permite emularea unor seturi diferite de instrucțiuni ale altor calculatoare (cu lungimea cuvîn-

tului variabilă de la 8 la 32 biţi, modular pe 8 biţi). În varianta sa iniţială, produsă în serie, minicalculatorul **I-100** are lungimea cuvîntului de instrucţiune pe 16 biţi şi este compatibil cu minicalculatorul **PDP-11/34** al firmei **DEC**, SUA.

Se utilizează microprogramarea pe orizontală, lungimea cuvîntului de microinstrucţiune fiind de 32 de biţi, cu un timp de execuţie de 200 nsec. Memoria de microprograme este formată din :

— o memorie fixă, de tip ROM, bipolar, cu capacitatea maximă de 2 cuvinte (de 32 biţi) şi timp de acces 50 nsec.
— o memorie dinamică, de tip RAM bipolar, cu capacitate maximă de 6 cuvinte (de 32 biţi) şi timp de acces 70 nsec.

În memoria de microprograme, 2 Kcuvinte sînt rezervate pentru emularea setului de instrucţiuni, restul fiind disponibile utilizatorilor (care îşi pot încărca propriile microprograme prin intermediul cititorului de bandă perforată sau unităţii de bandă magnetică). Pentru emularea setului de instrucţiuni **PDP-11** există şi un suport hardware adecvat : placa emulatoare **EMU-100**, care asistă microprogramele, sporind viteza de execuţie a acestora.

Elementele componente funcţionale ale minicalculatorului **I-100** sînt organizate în jurul a două magistrale interne de comunicaţie :

— **INTERBUS :**
  — asigură schimbul de informaţii între Unitatea Centrală şi echipamentele periferice ;
  — permite transmiterea de comenzi, stări, date şi vectori de întrerupere ;
  — conectează, opţional, memoria ;
  — conţine 72 de linii, din care 16 sînt pentru date şi 18 pentru adresare (a maximum 256 Ko) ;
  — permite conectarea a maximum 4 Unităţi Centrale ;
  — viteza de transfer a informaţiilor : 5Mo/sec.
— **MEMOBUS :**
  — asigură schimbul de informaţii între memorie, Unitatea Centrală şi echipamentele periferice funcţionînd în regim de acces direct la memorie **(DMA)** ;
  — reprezintă o variantă simplificată a **INTERBUS**-ului ;
  — conţine 49 de linii, din care 16 sînt pentru date şi 21 pentru adresare (a maximum 2 Mo) ;
  — permite conectarea de memorii cu acces dual ;
  — viteză de transfer a informaţiilor 5 Mo/sec.

Unitatea Centrală, microprogramată, este cuplată atît cu **INTERBUS**-ul cît şi cu **MEMOBUS**-ul prin intermediul unor registre de adresare **(ADR)** şi de date **(IDR/ODR)**.

Blocurile funcţionale ce fac parte din Unitatea Centrală a minicalculatorului **I-100**, corespunzătoare celor 13 plachete imprimate, sînt :

— Unitatea aritmetică şi logică **(ALU)**, ce execută operaţii aritmetice şi logice ;
— Decodificatorul de instrucţiuni **(DEC)** ;
— Placa emulatoare **(EMU)**, ce asistă microprogramele la implementarea unor instrucţiuni ;
— Blocul de control secvenţe, de generare ceasuri şi semnale de comandă **(SEQ)** ;

— Memoria de control microprograme (**ROM**) ;
— Extensia memoriei de control (**MPM**) ;
— Registrele de microinstrucțiuni (**MIR**) ;
— Registrul de instrucțiuni (**IR**) ;
— Blocul de întreruperi și condiții ;
— Unitatea de relocare și protecție a memoriei (**MMU**) ;
— Blocul de control **INTERBUS** (**IBS**) ;
— Blocul de control **MEMOBUS** (**MBS**) ;
— Blocul de control panou (**CSL**) ;
— Panoul de comandă (**FPB**).

Sertarul Unității Centrale poate conține 28 sau 44 de plachete. Pentru un număr mai mare de plachete se poate utiliza un sertar suplimentar.

Unitatea Centrală are următoarele caracteristici generale :

— implementează un set de instrucțiuni, cu unul sau doi operanzi, compatibil cu setul minicalculatorului **PDP-11/34** ;
— implementează, opțional, un set extins de instrucțiuni (**EIS**), ce asigură înmulțire și împărțire prin hardware ;
— implementează, opțional, un set de instrucțiuni în virgulă flotantă scurtă (**FIS**) ;
— asigură execuția a cca 400000—450000 operații/sec ;
— conține 9 registre generale, ce pot fi folosite ca acumulatori, registre index, registre bază sau indicatori de stivă ;
— asigură 8 moduri generale de adresare + 4 folosind contorul de instrucțiuni ;
— asigură facilități hardware pentru implementarea stivelor ;
— asigură un sistem de întreruperi vectoriale cu 4 nivele de priorități ;
— asigură acces direct la memorie (**DMA**) pentru echipamentele periferice cu viteză mare de transfer a datelor (acestea fiind conectate și la **INTERBUS** și la **MEMOBUS**) ;
— asigură salvarea automată a contorului de instrucțiuni (**PC**) și a stării program (**PS**), la apariția unor întreruperi (hardware sau software) ;
— detectează prin mijloace hardware căderile și fluctuațiile tensiunii de alimentare ;
— permite adresarea memoriei la nivel de octet (8 biți) și cuvînt (16 biți) ;
— permite adresarea directă a unei memorii de 32 Kcuvinte (de 16 biți) ;
— permite relocarea și protecția memoriei, cu extinderea adresării la 128 Kcuvinte (din care ultimele 4 Kcuvinte reprezintă registre de control și stare ale dispozitivelor periferice) ;
— permite două moduri de operare : *Sistem (Kernel)* și *Utilizator (User)*, cu un registru de stivă separat pentru fiecare mod ;
— asigură relocarea și protecția memoriei prin 2 seturi de cîte 8 registre de pagină (cîte un set pentru fiecare mod de operare) ; lungimea paginii este cuprinsă între 32 și 4 096 cuvinte ;
— asigură un emulator de consolă pe bandă perforată, casetă magnetică sau placă ROM.

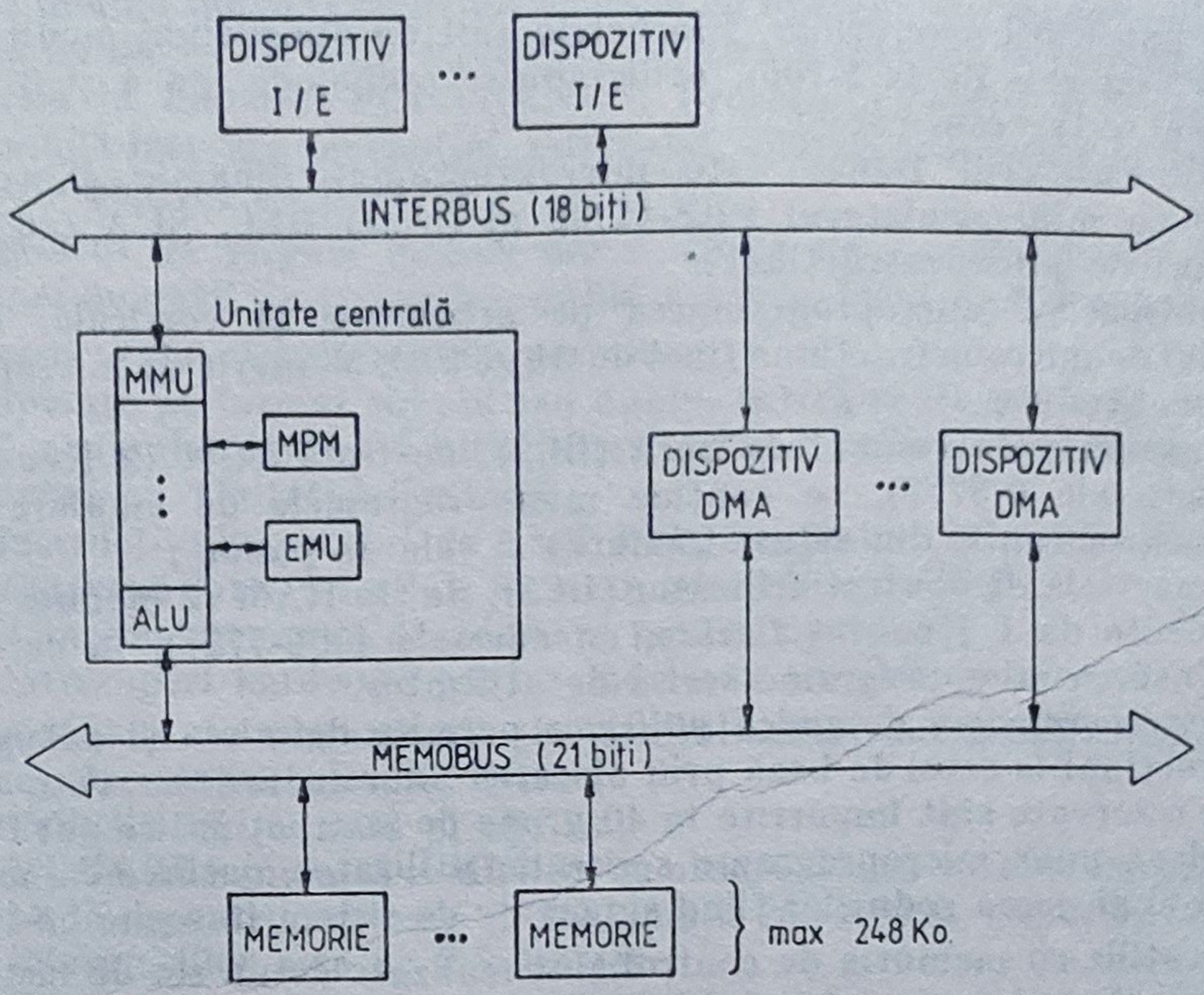


Fig. 1.1. Schema-bloc simplificată a minicalculatorului **I-100**

Minicalculatorul **I-100** este echipat cu două tipuri de memorii operative :

— memorie cu ferite tip AMPEX, identică cu cea folosită la sistemele **FELIX C-512/C-1024**, cu capacitatea minimă de 8 Kcuvinte (a 18 biți) și maximă de 32 Kcuvinte, timp de acces 400 nsec și ciclu de memorie de 800 nsec.

— memorie semiconductoare MOS, constituită în tehnologia specifică **I-100**, cu module de 32 Ko, 64 Ko și 128 Kocteți.

Memoria asigură controlul de paritate per octet (8 biți).

Minicalculatorul **I-100** folosește o gamă largă de echipamente periferice, dispunînd de cuploare specifice pentru fiecare tip de dispozitiv periferic (pe una sau mai multe plăci) : ceas de timp real programabil ; consolă operator cu interfață asincronă ; cititor perforator de bandă de hîrtie ; cititor de cartele ; imprimantă ; casetă magnetică dual ă ; disc cartridge de 2,5 și 5 Mo ; disc amovibil de 50 Mo ; bandă magnetică 1 600 bpi cu formatter ; interfață asincronă pentru o linie ; multiplexor asincron pentru 16 linii ; interfață sincronă pentru o linie.

Fig. 1.1 reprezintă schema-bloc simplificată a minicalculatorului **I-100**.

1.2.2. **Minicalculatorul I-102F**

Tehnologic, minicalculatorul **I-102F** utilizează în mare măsură componente electronice rapide tip Schottky, precum și circuite MSI și LSI. Unitatea Centrală rapidă, proiectată cu o structură microprogramată bazată pe circuite integrate pe scară largă de tip AM 2909, folosește tehnici speciale de

execuţie paralelă a operaţiilor. Plachetele sînt de dimensiuni medii (identice ca format cu cele de la **I-100**), utilizîndu-se tehnologia cu 4 şi 6 straturi; fundul de sertar este cablat.

Minicalculatorul **I-102F** este microprogramat dinamic şi compatibil software cu minicalculatorul **PDP-11/60** al firmei **DEC**, SUA (cu excepţia formatului de microinstrucţiune):

Se utilizează microprogramarea pe orizontală şi verticală, lungimea cuvîntului de microinstrucţiune fiind de 64 de biţi. Memoria de microprograme este formată din :

— memoria de control de bază (**BCS**), de tip PROM (2 Kcuvinte, cu adresele 0-3777), ce conţine microprogramele de emulare (pentru instrucţiunile din setul standard) şi cele de panou ;
— memoria de control utilizator (**UCS**), de tip RAM (2 module independente de 1 Kcuvînt fiecare, cu adresele 4000-7777), în care se pot încărca microprograme scrise de utilizator.

Microprogramarea dinamică utilizator permite definirea şi adăugarea de noi instrucţiuni la setul de bază prin alocarea unor coduri rezervate acestora. Codurile rezervate sînt împărţite în 40 grupe de instrucţiuni ce pot fi alocate independent unor microprograme scrise de utilizator, încărcarea microprogramelor şi alocarea codurilor fiind suportate de sistem în regim on-line.

Operaţiile cu memoria de control sînt realizate de un set de instrucţiuni (**XFC**), specific minicalculatorului **I-102F**, ce nu se pot executa decît în mod Kernel.

Microprogramarea dinamică a minicalculatorului **I-102F** este facilitată de o serie de interfeţe software :

— un microasamblor, **MIC102F**, ce permite scrierea microprogramelor utilizator într-un limbaj adecvat ;
— un driver în sistemul de operare (**ZFDRV**), care verifică validitatea operaţiilor cu memoria de control, asigură iniţializarea acesteia şi permite facilităţi de protecţie între utilizatori ;
— un încărcător on-line de microprograme, **MPL102F**, ce asigură încărcarea microprogramelor utilizator şi alocarea/dezalocarea de coduri de instrucţiuni.

Pentru mărirea vitezei de execuţie a minicalculatorului **I-102F** se utilizează o memorie tampon specială, ultrarapidă, (utilizînd circuite bipolare), numită *cache*, plasată între memoria principală şi unitatea centrală, ce menţine o copie a unor porţiuni de memorie selectate recent, pentru a asigura un acces ulterior mai rapid la date şi instrucţiuni.

Memoria cache are capacitatea de 2 Kocteţi, organizaţi în 1024 cuvinte de 27 biţi fiecare, şi un timp de acces de 150 nanosecunde (ciclu de memorie mai mic decît 250 nsec). Ea este fizic localizată în Unitatea Centrală, este sincronizată cu procesorul de bază şi elimină timpii lungi de acces şi transmisie pe magistrală, asociaţi cu memoria principală. Dispozitivele **DMA** nu utilizează memoria cache.

Utilizarea memoriei cache este transparentă utilizatorului, reactualizarea conţinutului făcîndu-se automat, dinamic.

În scopul măririi vitezei de execuţie a programelor cu un volum mare de calcule numerice, minicalculatorul **I-102F** poate avea opţional un *Procesor de virgulă mobilă* (**FPP**), ce implementează setul complet (46) de instrucţiuni

de virgulă mobilă lungă (executînd operații cu operanzi de lungime simplă — 32 biți sau dublă — 64 biți), asigurînd o eficiență maximă la execuția programelor scrise în limbajul FORTRAN-77. Procesorul Central și Procesorul de virgulă mobilă lucrează secvențial, intercalat, ambele procesoare avînd aceeași memorie de control de bază.

Procesorul de virgulă mobilă are o viteză de execuție foarte ridicată, durata unei operații de înmulțire, dublă precizie, fiind de 5,4 μ secunde.

Procesorul de virgulă mobilă permite :

— operare pe format simplu sau dublu, întreg scurt sau lung ;
— 6 acumulatori de 64 biți, primii 4 fiind utilizați și pentru comunicarea cu Unitatea Centrală ;
— instrucțiunile setului de bază **FPP**, conținînd operații aritmetice și conversii între diferitele formate adresabile de **FPP** ;
— 8 moduri generale de adresare pentru operanzi ;
— întreruperi hardware pentru indicarea erorilor de operare.

Magistralele de comunicații ale minicalculatorului I-102F sînt identice cu cele de la minicalculatorul I-100 ;

— **INTERBUS** (numită și **IBUS**) ;
— **MEMOBUS** (numită și **MBUS**), cu observația că aceasta conține numai 18 linii pentru adresare (a maximum 256 Ko) și viteza de transfer a informațiilor este de 2,5 Mo/sec.

Blocurile funcționale ce fac parte din Unitatea Centrală a minicalculatorului **I-102F**, corespunzătoare celor 7 plachete imprimate, sînt :

— Unitatea aritmetică și logică (**ALU**) ;
— Controlul de secvențe (**SEQ**) ;
— Logica de control (**CTL**) ;
— Memoria de control microprograme (**ROM**) ;
— Memoria cache (**CCH**) ;
— Unitatea de relocare și protecție a memoriei (**MMU**) ;
— Controlorul de intrare/ieșire (**IOC**).

În blocul Unității Centrale se mai găsesc 2 plachete :

— Bootstrap și terminator de **BUS** (**BOT**) ;
— Panoul de comandă (**MCSL**).

Procesorul de virgulă mobilă conține 3 plachete (cu circuite tip **MSI**) :

— Controlul de **MACRO** și mantisa inferioară (**MCR**) ;
— Controlul mantisei și mantisa superioară (**MCL**) ;
— Exponent și condiții (**BCD**) ;

Memoria de control utilizator (**UCS**) este conținută pe o plachetă. Sertarul Unității Centrale poate conține 28 de plachete. Pentru un număr mai mare de plachete se poate utiliza un sertar suplimentar.

Unitatea centrală a minicalculatorului **I-102F** are caracteristici generale similare cu cele ale minicalculatorului **I-100**, cu următoarele observații :

— implementează un set de instrucțiuni, cu unul sau doi operanzi, compatibil cu setul minicalculatorului **PDP-11/60** ;
— include standard setul extins de instrucțiuni (**EIS**) ;
— implementează, opțional, setul de instrucțiuni în virgulă flotantă lungă (**FPP**) ;

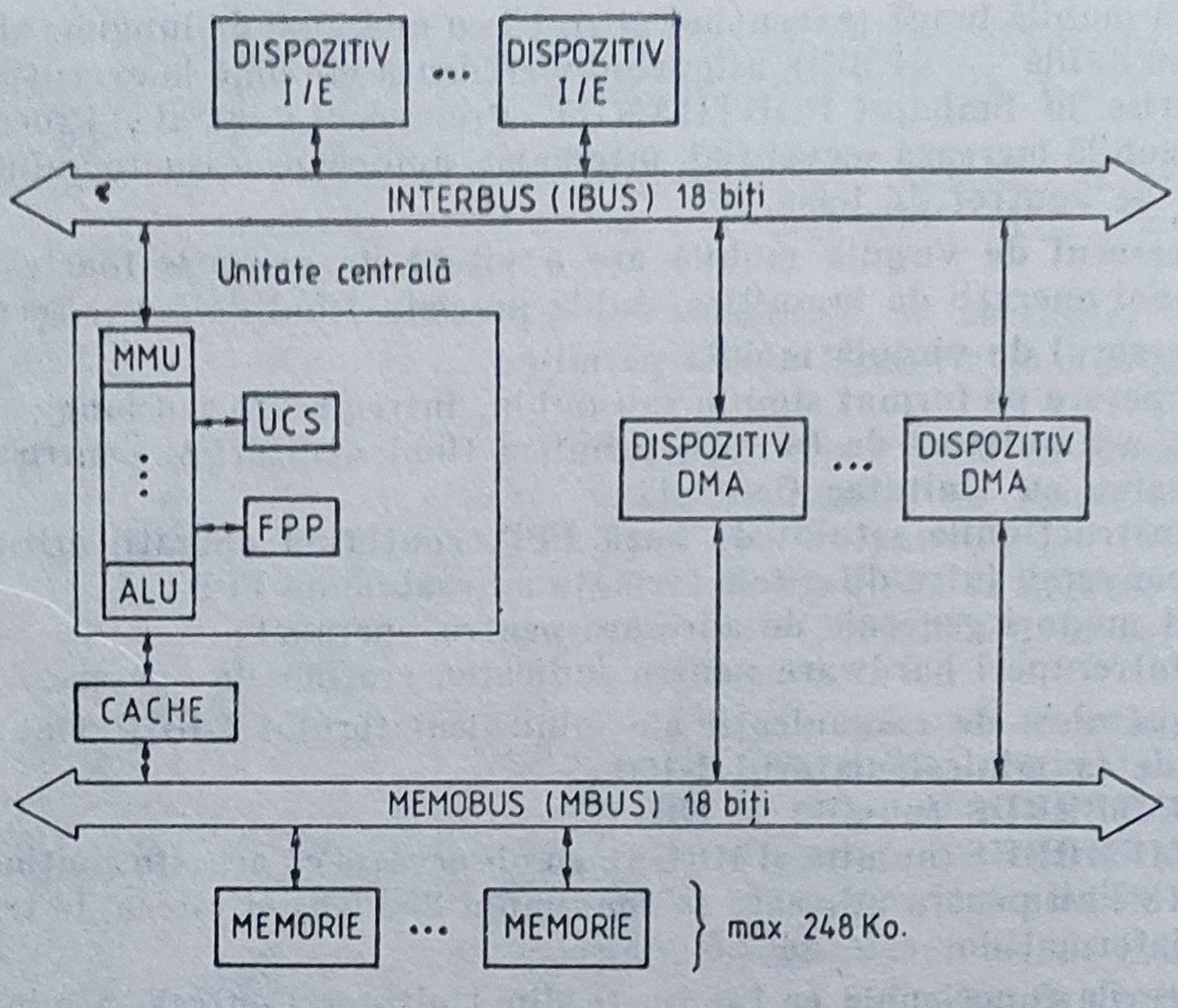


Fig. 1.2. Schema-bloc simplificată a minicalculatorului I-102F

— asigură execuția a cca 2 500 000 operații simple (registru-registru)/secundă și a unei medii de 800 000—1 000 000 operații cu acces la memorie/secundă ;

— unitatea de relocare și protecție a memoriei asigură acces la spații instrucțiuni (**I**) și date (**D**) separate, pentru fiecare spațiu existînd 2 seturi de cîte 8 registre de pagină (corespunzătoare modurilor de operare sistem și utilizator).

Minicalculatorul **I-102F** este echipat cu plachete de memorie de tip **MOS**, cu capacitatea de 32 Ko și 128 Kocteți.

Minicalculatorul **I-102F** folosește o gamă largă de echipamente periferice, dispunînd de toate cuploarele specifice acestora de pe minicalculatorul **I-100**. În plus față de aceste echipamente, s-au mai realizat cuploare specifice pentru : multiplexor asincron pentru 8 căi (**AMI**) ; procesor de comunicații (**ASYX**) ; diverse cuploare și interfețe pasive de proces (convertoare A/N și N/A), etc.

Fig. 1.2 reprezintă schema-bloc simplificată a minicalculatorului **I-102F**.

1.2.3. Minicalculatorul I-102 F cu extensie de memorie

Acest minicalculator reprezintă o extensie funcțională a modelului **I-102F**. în sensul creșterii posibilităților de acces la memorie de la 248 Ko la 3 Mo, păstrîndu-se toate caracteristicile arhitecturale de bază ale minicalculatorului **I-102F**.

Precizările anterioare făcute pentru minicalculatorul **I-102F** rămîn valabile și pentru minicalculatorul **I-102F** cu memorie extinsă, cu următoarele observații :

— memoria adresabilă a crescut de la 248 Ko la 3 Mo (limitare impusă de capacitatea sertarului principal) ;
— Unitatea de relocare și protecție a memoriei a fost reproiectată, incluzînd un dispozitiv de translatare (mapare) a adreselor (**BUS MAP**), pentru a asigura accesul dispozitivelor **DMA** la memoria extinsă ;
— în Unitatea Centrală a apărut o plachetă nouă, corespunzătoare translatării (mapării) adreselor pe magistrala internă ;
— liniile de adresă aparținînd magistralei **MEMOBUS** au fost extinse de la 18 la 22, pentru a permite adresarea a maximum 4 Mo memorie ;
— panoul de comandă specific minicalculatoarelor **I-100** și **I-102F** a fost înlocuit cu altul mai simplu și cu un număr redus de comutatori ; microprogramul de panou conține un subsistem de depanare incorporat ;
— memoria utilizată este de tip **MOS**, folosind circuite de 16 Kbiți, cu capacitatea de 256 Kocteți/plachetă ;
— s-au reproiectat o parte din cuploarele dispozitivelor periferice în scopul micșorării numărului de plachete/cuplor sau aglomerării mai multor cuploare pe aceeași plachetă.

În plus față de aceste reproiectări, s-a mai realizat un cuplor specific pentru discul flexibil dublă densitate.

Fig. 1.3 reprezintă schema-bloc simplificată a minicalculatorului **I-102F** cu extensie de memorie.

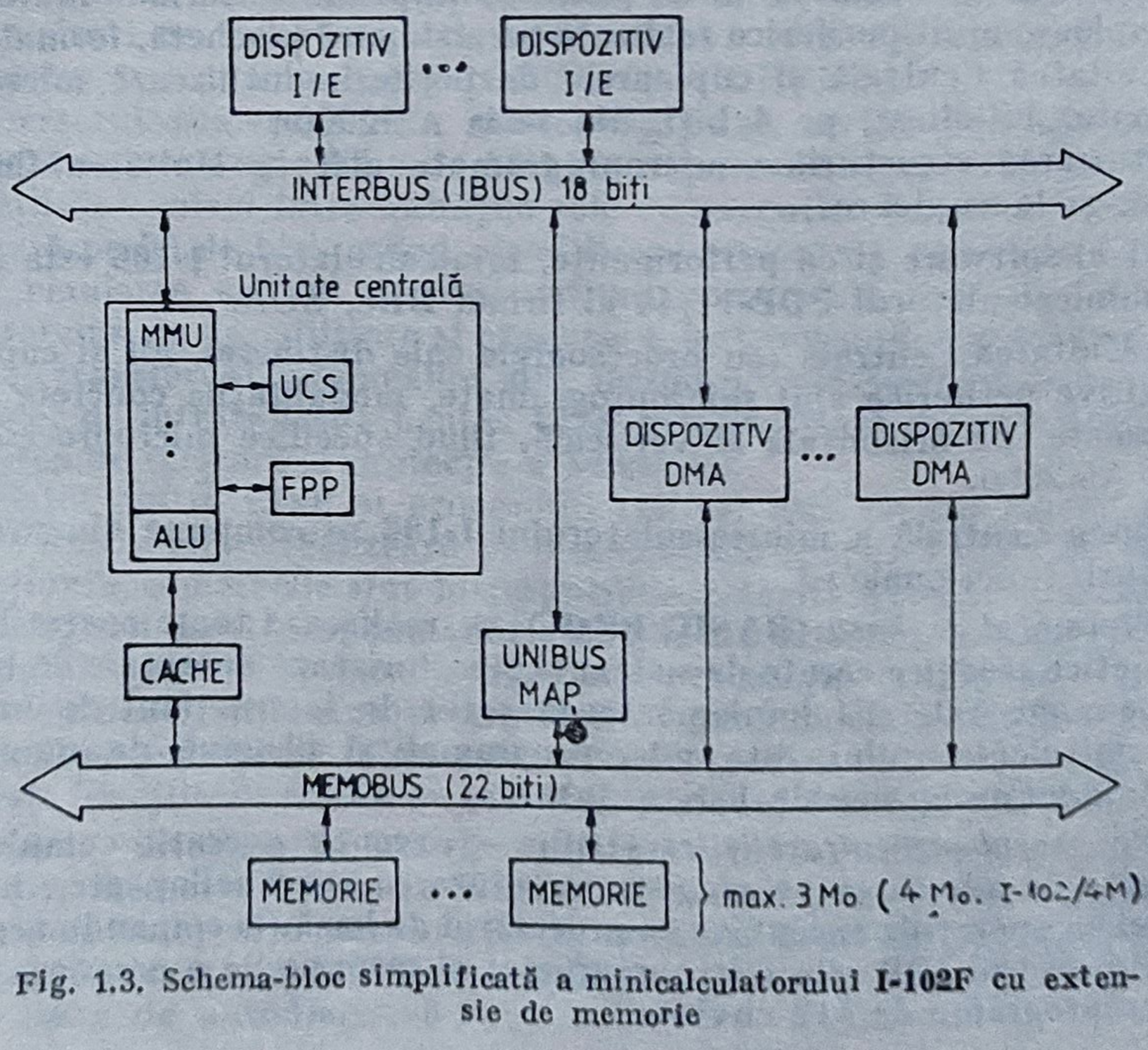


Fig. 1.3. Schema-bloc simplificată a minicalculatorului I-102F cu extensie de memorie

1.2.4. Minicalculatorul I-102/4M

Acest minicalculator reprezintă o reproiectare tehnologică a modelului **I-102F** cu extensie de memorie, în sensul creşterii posibilităţilor de acces la memorie de la 3 Mo la 4 Mo şi micşorării gabaritului (un singur sertar cu 28 plachete), păstrîndu-se caracteristicile arhitecturale de bază ale minicalculatorului **I-102F.**

Principalele modificări tehnologice aduse acestui minicalculator sînt:

— reproiectarea plachetelor de pe 6 pe 4 straturi;
— micşorarea numărului de plachete la cuplorul de bandă magnetică 800/1 600 bpi;
— memoria utilizată este de tip **MOS**, folosind circuite de 64 kbiţi, cu capacitatea de 1 Mo/plachetă;
— s-au reproiectat o parte din cuploarele simple pentru mărirea gradului de integrare.

Schema-bloc simplificată a minicalculatorului **I-102/4M** este identică cu cea a minicalculatorului **I-102F** cu extensie de memorie (vezi fig. 1.3).

1.2.5. Minicalculatorul I-106

Minicalculatorul **I-106**, conceput ca un sistem de calcul de mare productivitate pentru aplicaţii diverse, este realizat utilizînd o tehnologie avansată, mult diferită de cea a celorlalţi doi membri ai familiei, **I-100** şi **I-102F**:

— Unitatea Centrală pe două plachete imprimate, format mare;
— cuploare multiperiferice realizate pe o singură plachetă, format mare;
— Unitatea Centrală şi cuploarele de periferice utilizează microprocesoare „bit-slice", pe 4 biţi, din seria AM2900;
— utilizarea structurilor microprogramate atît la Unitatea Centrală, cît şi la cuploare.

La nivel software şi ca performanţe, minicalculatorul **I-106** este compatibil cu minicalculatorul **PDP-11/44** al firmei **DEC**, SUA.

Atît Unitatea Centrală (cu procesoarele sale distincte), cît şi cuploarele de dispozitive periferice sînt microprogramate, modalităţile concrete de microprogramare, pe orizontală şi verticală, fiind specifice microprocesoarelor din seria AM2900.

Unitatea Centrală a minicalculatorului **I-106** se compune din următoarele blocuri funcţionale:

— *Procesorul de bază* **(BASIC PROC)** — realizează toate operaţiile aritmetice şi logice cerute de sistem şi monitorizează operaţiile de transfer pe magistrale. El implementează setul de instrucţiuni de bază ale minicalculatorului, este microprogramat şi dispune de o memorie de microprograme de 1 Kcuvînt.
— *Procesorul de intrare/ieşire* **(IOP)** — execută operaţii complexe legate de adresarea pe magistrale, aducerea operanzilor, etc.. în paralel cu operaţiile executate de procesorul de bază (la comanda acestuia). Procesorul **IOP** este microprogramat şi dispune de o memorie de microprograme de 512 cuvinte.

Procesorul **IOP** înglobează și *memoria cache* (**CCH**), care are o capacitate de 4 Kocteți, organizați în 2048 cuvinte de 31 biți fiecare, și un timp de acces de 150 secunde.

— *Dispozitivul de relocare și protecție a memoriei* (**MMU**) ce asigură conversia adresei virtuale pe 16 biți în adresă fizică de memorie pe 22 biți și protecția accesului la memorie. Relocarea și protecția se fac separat pentru spațiile de date (**D**) și instrucțiuni (**I**), pentru fiecare din cele trei moduri de lucru ale Unității Centrale (*Sistem*, *Supervizor* și *Utilizator*).

— *Dispozitivul de mapare a magistralei de dispozitive de intrare/ieșire* (**BUS MAP**), ce asigură conversia adresei de 18 biți de pe magistrala de periferice (**SBUS**) în adresa de 22 biți de pe magistrala de memorie, pentru accesele **DMA** ale cuploarelor.

Procesorul de virgulă mobilă (**FPP**), destinat execuției eficiente a operațiilor cu numere reale în precizie simplă (32 biți) sau dublă (64 biți), execută același set de instrucțiuni (46) cu cel de la minicalculatorul **I-102F**, dar diferă din punct de vedere al performanțelor, structurii interne și al tehnologiei (folosind circuite electronice LSI).

Arhitectura procesorului de virgulă mobilă este complet separată de cea a procesorului de bază, cele două procesoare, microprogramate, avînd memorii de control microprograme diferite. Transferul de date cu unitatea aritmetică a procesorului de bază se face prin intermediul unei magistrale bidirecționale **ABUS**, de 16 biți. Durata unei operații de înmulțire, dublă precizie, este de aproximativ 10 μsecunde.

Minicalculatorul **I-106** conține de asemenea un *procesor comercial* (**CISP**) opțional, destinat efectuării operațiilor în aritmetică zecimală și a operațiilor pe șiruri de caractere, asigurînd astfel o eficiență maximă la execuția programelor scrise în limbajul COBOL sau COBOL-81.

Procesorul comercial este microprogramat, avînd o structură internă organizată pe un singur **BUS** de 16 biți, ce permite transferuri de operanzi și informații de control între unitățile sale de execuție. Microinstrucțiunea este organizată pe 56 de biți, avînd o codificare orizontală cu cîmpuri suprapuse.

În emularea setului de instrucțiuni comerciale, procesorului **CISP** îi revine sarcina de decodificare și execuție a instrucțiunilor și control al operațiilor de transfer al operanzilor din memoria minicalculatorului. Execuția operațiilor de intrare/ieșire pentru **CISP** este realizată de procesorul de bază și unitatea de relocare și protecție a memoriei.

Relația dintre **CISP** și procesorul central este de tip master-slave, iar legătura dintre ele se realizează pe o magistrală specifică (**ABUS**), de 16 biți. Instrucțiunile comerciale sînt întreruptibile datorită duratei mari de execuție a lor, lungimii mari a operanzilor (pînă la 64 kocteți), și timpului necesar transferului operanzilor între memorie și procesorul comercial.

Un element important în arhitectura minicalculatorului **I-106** îl reprezintă *Consola inteligentă (Procesorul de control și diagnosticare)* (**CI**), care îndeplinește funcțiile de panou de comandă și modul de diagnosticare (depanare).

Consola inteligentă constă dintr-un microcalculator (tip **Z80**) cu 6 Kocteți de memorie RAM, 12 Kocteți de memorie EPROM, un disc floppy (simplă densitate) cu două unități, un terminal și o linie pentru depanare de la distanță. Memoria EPROM conține :

— teste de autodiagnoză ;

— teste pentru diagnosticarea procesorului de bază ;
— sistemul de operare al consolei (**COS**) ;
— programe de încărcare de pe anumite dispozitive periferice.

Consola inteligentă servește ca :

— *terminal (consolă operator)* a sistemului de operare ce rulează pe minicalculatorul **I-106 (MIX-PLUS)**. Acestuia îi este asociat un cuplor asincron cu o cale și un ceas de linie.
— *consolă a sistemului (panou)*, utilizată pentru comanda minicalculatorului (inițializare, funcții de panou, actualizare registru comutatori, încărcare a sistemului de operare **MIX-PLUS**) ;
— *consolă de diagnosticare*, controlînd Unitatea Centrală și cuploarele de periferice prin intermediul unei *magistrale de diagnosticare* (**DGB**) în puncte de control, special prevăzute în blocurile funcționale și cuploarele de dispozitive.

Consola de diagnosticare permite :

— vizualizarea la cerere a stării calculatorului ;
— rularea, chiar în consola inteligentă, a unor programe de text (diagnoză) încărcabile de pe discul flexibil asociat ;
— facilități de depanare de la distanță.
— *cuplor de disc flexibil* (simplă sau dublă densitate) pentru minicalculatorul **I-106**.

Consola inteligentă servește ca interfață între operator și minicalculator, oferind utilizatorului metode de întreținere și flexibilitate în operare. Limbajul de comandă al consolei este similar cu limbajul de comandă al sistemului de operare (**COS**).

Memoria principală (**MEM**) a minicalculatorului **I-106** este organizată ca un controler specific, la care se pot atașa 1—4 module de memorie. Memoria modulară, de tip **MOS**, este prevăzută cu circuite detectoare și corectoare de erori (**ECC**). Capacitatea unui modul este de 512 Ko (cu circuite de 16 Kbiți), 1Mo sau 2 Mo (cu circuite de 64 Kbiți). Timpul de acces este de aproximativ 750ns la citire și 250 ns la scriere.

Magistralele de comunicații ale minicalculatorului **I-106** sînt următoarele :

— **SBUS :** magistrală de conectare a Unității Centrale cu dispozitivele de I/E și memoria, cu 16 linii de date și 18 de adresă ;
— **ABUS :** magistrală internă de comunicație între Procesorul de bază și procesoarele **FPP** și **CISP**, cu 16 linii de date/adresă ;
— **MA :** magistrală internă de acces la memorie, cu 22 linii de adresă la care se conectează Unitatea Centrală și dispozitivul de mapare ;
— **DGB :** magistrală de diagnosticare, la care sînt conectate punctele de diagnosticare din Unitatea Centrală și cuploarele microprogramate ; magistrala de diagnosticare este monitorizată de consola inteligentă și conține 8 linii de adresă/date.

Unitatea centrală a minicalculatorului **I-106** are caracteristici generale similare cu cele ale minicalculatoarelor din familia **INDEPENDENT**, cu următoarele observații :

— implementează un set de instrucțiuni, cu unul sau doi operanzi, compatibil cu setul minicalculatorului **PDP-11/44** ;

— implementează standard setul extins de instrucțiuni (**EIS**).
— implementează, opțional, setul de instrucțiuni în virgulă flotantă lungă (**FPP**) și setul de instrucțiuni comercial (**CIS**) :
— asigură facilități de punere la punct și depanare prin :
  — existența unei magistrale de diagnoză ;
  — consola inteligentă.
— include caracteristici hardware pentru implementarea sistemelor de operare multiutilizator, de mare performanță (**MIX-PLUS**) :
  — unitate de relocare și protecție a memoriei ce asigură conversia adresei virtuale de 16 biți în adresă fizică de 22 biți ;
  — spații separate de instrucțiuni (**I**) și date (**D**) ;
  — trei moduri de lucru cu relocare distinctă (*Sistem, Supervizor, Utilizator*) ;
  — instrucțiuni speciale, destinate implementării eficiente a unor mecanisme din sistemele de operare.
  — are o viteză de execuție de cca 1 milion operații/secundă.

Minicalculatorul **I-106** folosește o gamă largă de echipamente periferice, dezvoltate în jurul a două tipuri de cuploare universale microprogramate :
— cuplor universal de discuri magnetice, ce permite cuplarea a : 4 unități de discuri de 50 Mo (DD), 8 unități de discuri de 2,5 Mo (DK), 4 unități de discuri de 10 Mo (DL), 4 unități de discuri de 40 Mo (DP), 2 unități de discuri de 200 Mo (DB) sau 2—4 unități discuri Winchester de mare capacitate ;
— cuplor universal periferice lente, ce permite cuplarea unei imprimante (**LP**), unui cititor de cartele (**CR**), unui cititor/perforator bandă de hîrtie (**PR/PP**) sau 8 linii asincrone seriale (tip **DL, DZ**).

Alte echipamente ce se pot cupla la **I-106** sînt :
— bandă magnetică 800/1 600 bpi (**MM**) sau 1600 (**MS**) sau streaming (**MF**) ;
— procesor de comunicații tip **KMC (ASYX)** ;
— cuplor de rețea locală tip Ethernet (**DEUNA**), etc.

Fig. 1.4 reprezintă schema-bloc simplificată a minicalculatorului **I-106.**

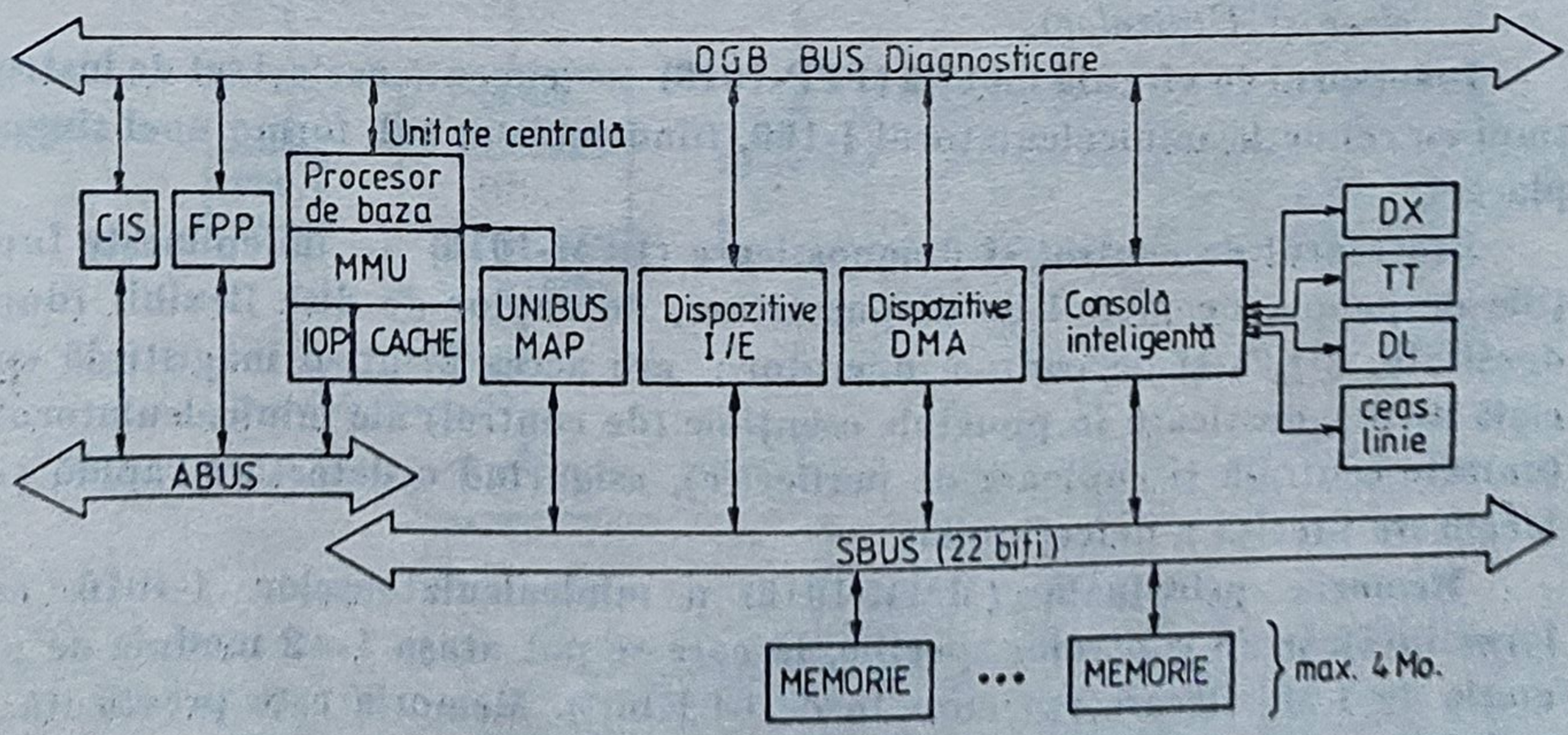


Fig. 1.4. Schema-bloc simplificată a minicalculatorului I-106

1.2.6. Minicalculatorul I-1016

Minicalculatorul **I-1016**, conceput în special a fi utilizat în configurații specifice locurilor de muncă industriale, constituie o reproiectare majoră a minicalculatorului **I-106**, utilizînd o tehnologie adecvată :

— plachete imprimate realizate cu 4 straturi ;

— unitate centrală realizată pe 2 plachete imprimate ;

— utilizarea structurilor microprogramate (cu microprocesoare „bit-slice", seria AM2900), atît pentru unitatea centrală cît și pentru cuploarele de periferice ;

— utilizarea a numeroase circuite cu logică programată din familiile TTL și MOS.

La nivel software, minicalculatorul **I-1016** este compatibil cu minicalculatorul **PDP-11/44** al firmei **DEC**, S.U.A.

Unitatea Centrală a minicalculatorului **I-1016** se compune din următoarele blocuri funcționale :

— *Procesorul de bază* **(CPU-1016)** — execută setul de bază al instrucțiunilor calculatorului, este microprogramat și dispune de o memorie de microprograme de 1 Kcuvînt ;

— *Procesorul de intrare/ieșire* **(IOP-1016)** — execută operațiile complexe legate de adresarea pe magistralele interne, în paralel cu operațiile executate de procesorul de bază (la comanda acestuia). Procesorul este microprogramat, înglobînd și memoria *cache* care are o capacitate de 4 Kocteți ;

— *Dispozitivul de relocare și protecție a memoriei* **(MMU-1016)** — care asigură conversia adresei virtuale pe 16 biți în adresă fizică de memorie pe 22 biți și protecția accesului la memorie. Relocarea și protecția se face separat pentru spațiile de date **(D)** și de instrucțiuni **(I)**, pentru fiecarefdin cele 3 moduri de lucru ale unității centrale *(Sistem, Supervizor* și *Utilizator)*.

Procesorul de virgulă mobilă (FPP-1016) — execută același set de instrucțiuni cu cel de la minicalculatorul **I-106**, fiind realizat sub forma unei singure plachete.

Procesorul de control și diagnosticare (DCM-1016) — îndeplinește funcțiile de panou, de modul de diagnosticare, de cuplor de disc flexibil (dublă densitate, $5^1/_4''$) și de consolă operator ; are acces printr-o magistrală specială de diagnosticare în punctele esențiale (de control) ale minicalculatorului (unitate centrală și cuploare de periferice), asigurînd o detectare rapidă și o localizare precisă a defecțiunilor.

Memoria principală (MMM-1016) a minicalculatoarelor I-1016 este formată dintr-un controlor specific, la care se pot atașa 1—2 module de memorie de 1 Mo fiecare (cu circuite de 64 Kbiți). Memoria este prevăzută cu cod detector de erori (simple și duble) și corector de erori simple.

Magistralele de comunicaţii ale minicalculatorului **I-1016** sînt următoarele :

— **S24 :** magistrală de conectare a Unităţii Centrale cu dispozitivele de I/E şi cu memoria principală ; este o magistrală asincronă pe 24 de biţi, cu linii comune pentru adrese (24) şi date (16) ;

— **FBUS :** magistrală internă de comunicaţie între Procesorul de bază şi Procesorul de virgulă mobilă ; este o magistrală pe 16 biţi de date/adrese ;

— **DGB :** magistrală de diagnosticare (pe 8 biţi), la care sînt conectate punctele de diagnosticare din Unitatea Centrală şi cuploarele microprogramate de periferice.

Unitatea Centrală a minicalculatorului **I-1016** are caracteristici similare cu cele ale minicalculatorului **I-106**, cu următoarele deosebiri :

— nu există procesorul comercial (**CISP**), datorită destinării industriale a minicalculatorului ;

— nu există dispozitivul de translatare a adreselor („*mapare*") a magistralei de dispozitive de intrare/ieşire (**BUS MAP**), accesul **DMA** al cuploarelor fiind realizat direct cu memoria principală (într-o manieră asemănătoare cu cea de la minicalculatoarele **CORAL-4030/4021**).

Minicalculatorul **I-1016** foloseşte o gamă limitată de echipamente periferice, cu funcţionare fiabilă în condiţii specifice mediilor industriale :

— cuplor şi unitate de disc magnetic Winchester (**WDC-1016**) ;

— cuplor şi unitate de bandă magnetică cu transfer continuu (**MTC-1016**) ;

— cuplor pentru 8 linii seriale asincrone şi pentru imprimantă (**ALC-1016**).

Ca opţiuni, se pot conecta :

— cuplor de reţea locală tip **ETHERNET** ;

— procesor grafic ;

— interfeţe de proces.

Schema-bloc simplificată a minicalculatorului **I-1016** este dată în fig. 1.5.

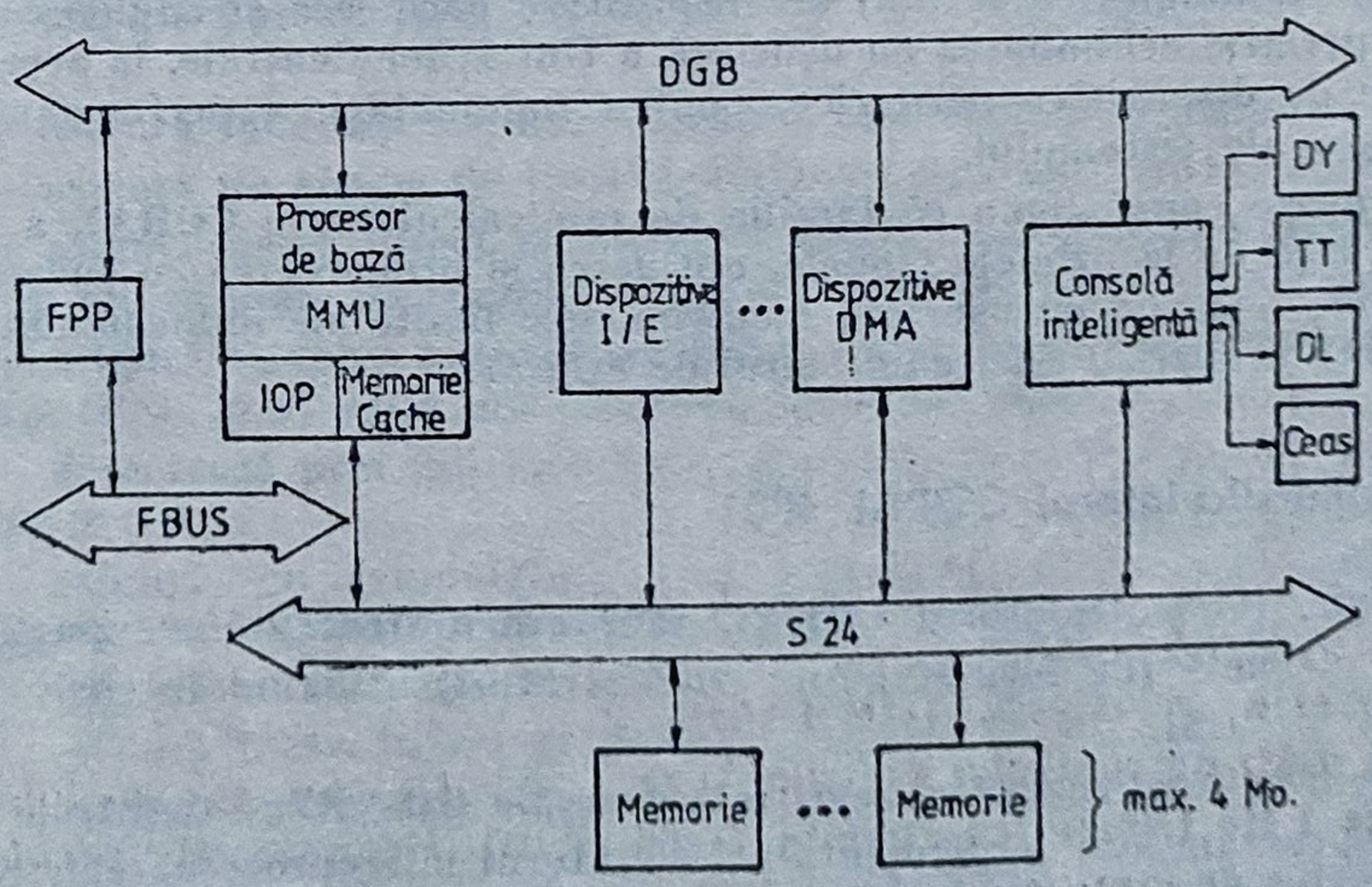


Fig. 1.5. Schema-bloc simplificată a minicalculatorului I-1016

1.3. Familia de minicalculatoare CORAL

Proiectată și fabricată la Întreprinderea de Calculatoare Electronice (ICE) București, familia de minicalculatoare **CORAL** include cinci tipuri de procesoare de bază, un mare număr de dispozitive periferice și suport software adecvat, reprezentînd un succes de performanță al tehnicii de calcul românești.

Procesoarele de bază din familia **CORAL** sînt următoarele :

— **CORAL 4001** și dezvoltarea sa **CORAL 4001A** ;
— **CORAL 4011** și dezvoltarea sa **CORAL 4011A** ;
— **CORAL 4030** ;
— **CORAL 4021 (DP21)**, cu dezvoltarea sa **CORAL 4051 (DP15)** ;
— **CORAL 8730** (minicalculator pe 32 de biți, ce nu face obiectul prezentei lucrări).

Arhitectura comună pe care se bazează familia **CORAL** o constituie sistemul de comunicare de tip **BUS**, între diversele module funcționale și dispozitivele periferice.

În ultimii zece ani, în paralel cu modernizarea continuă, prin folosirea componentelor electronice de cea mai bună calitate, compacte și fiabile (microprocesoare bit-slice, FPLA-uri, PROM-uri de mare capacitate), familia **CORAL** și-a păstrat atît compatibilitatea software, cît și cea hardware între diversele modele. Această modularitate permite obținerea cu ușurință a unei game largi de configurații, oferind și viteza cerută de aplicații din cele mai complexe.

Unitățile Centrale ale minicalculatoarelor **CORAL** sînt organizate pe una **(CORAL 4001(A), 40115)** sau două **(CORAL 4011(A), 4030, 4021)** plachete imprimate, format mare, în tehnologia pe 4 și 6 straturi. Fundul de sertar este realizat cu conexiuni imprimate, iar sursa unică de alimentare, cu comutație, este miniaturizată.

O caracteristică importantă a familiei de minicalculatoare **CORAL** o constituie și unicitatea setului de dispozitive periferice și cuploare atașate, ceea ce permite schimbarea cu ușurință a Unităților Centrale, la aceeași configurație de dispozitive periferice, pentru îmbunătățirea performanțelor de exploatare ale sistemului.

Se poate concluziona că familia de minicalculatoare **CORAL** a avut în vedere, încă de la primele modele, obținerea și promovarea cu succes a performanței, depășind cu ușurință tradiționalele limite ale minicalculatoarelor, în vederea acoperirii unei game largi de aplicații.

1.3.1. Minicalculatorul CORAL 4001

Tehnologic, minicalculatorul **CORAL 4001** are o structură compactă, pe o singură plachetă (de format mare, cu 6 straturi) utilizînd intensiv circuite integrate tip LSI.

Blocul aritmetico-logice, registrele generale **R0 ÷ R7** și registrele interne de microprogram sînt implementate cu ajutorul microprocesoarelor bit-slice, de tipul AM2901. Utilizînd eficient microprogramarea pe orizontală (cu lun-

gimea cuvîntului de microinstrucțiune de 64 biți), blocul aritmetico-logic realizează setul complet de instrucțiuni (inclusiv setul **EIS** — înmulțire și împărțire hardware) în 512 cuvinte.

Minicalculatorul **CORAL 4001** are lungimea cuvîntului de instrucțiune pe 16 biți și este compatibil cu minicalculatorul **PDP-11/04** al firmei **DEC**, SUA.

Elementele componente ale minicalculatorului **CORAL 4001** sînt organizate în jurul unei magistrale de comunicație unice, numită **BUS**, folosită pentru comunicarea între diversele module și dispozitive periferice. Unitatea Centrală, memoria și dispozitivele periferice folosesc un set unic de semnale pentru a comunica între ele. **BUS**-ul conține 16 linii pentru date și 16 pentru adresare (a maximum 32 Kcuvinte), viteza maximă de transfer a informațiilor fiind de 10 Mo/secundă.

Plachetele principale ce fac parte din configurația **CORAL 4001** sînt următoarele :

- **CP01** — Unitate Centrală pe 6 straturi ;
- **BC40** — Interfață panou, emulator consolă, registru eroare paritate și controler de **BUS**, pe 6 straturi ;
- **Panou** cu registre de adrese, date și comutatori.

Sertarul minicalculatorului poate conține 6, 12 sau 18 plachete.

Caracteristicile generale ale Unității Centrale sînt :

— implementează un set de instrucțiuni, cu unul sau doi operanzi, compatibil cu setul minicalculatorului **PDP-11/04** ;

— implementează standard setul extins de instrucțiuni (**EIS**) ;

— permite adresarea memoriei la nivel de octet (8 biți) și cuvînt (16 biți) ;

— permite adresarea directă a unei memorii de 28 Kcuvinte (de 16 biți), ultimele 4 Kcuvinte reprezentînd pagina externă ;

— asigură execuția a cca. 350 000 operații pe secundă ;

— conține 8 registre generale, ce pot fi folosite ca acumulatori, registre index, registre bază sau indicatori de stivă ;

— asigură 8 moduri generale de adresare + 4 folosind contorul de instrucțiuni ;

— asigură facilități hardware pentru implementarea stivelor ;

— asigură un sistem de întreruperi vectoriale cu 4 nivele de priorități ;

— asigură acces direct la memorie (**DMA**) pentru echipamentele periferice cu viteză mare de transfer al datelor (acestea fiind conectate la **BUS**) ;

— asigură salvarea automată a contorului de instrucțiuni (**PC**) și a stării program (**PS**), la apariția unor întreruperi (hardware sau software) ;

— detectează prin mijloace hardware căderile și fluctuațiile tensiunii de alimentare ;

— asigură, cu acumulatori, păstrarea conținutului memoriei *(battery backup)* în cazul căderilor de tensiune *(power fail)* ;

— asigură un emulator de consolă pe circuite ROM ;

— asigură încărcătoare primare pe circuite ROM ;

— asigură autotestare software a Unității Centrale, executată automat la pornirea calculatorului ;

— conține registru de comutatori hardware.

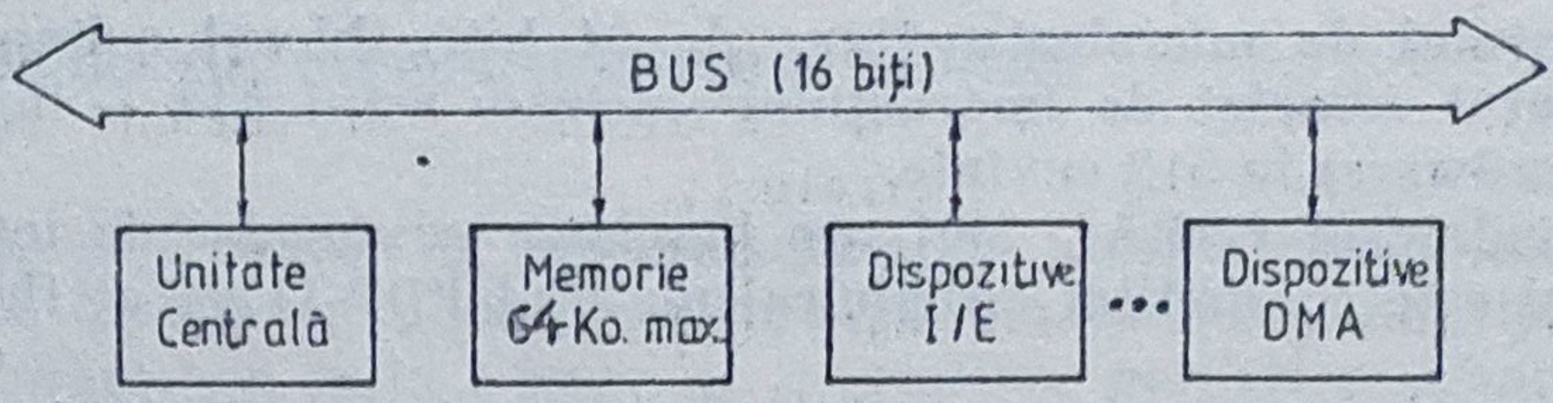


Fig. 1.6. Schema-bloc simplificată a minicalculatoarelor CORAL 4001 și 4001A

Minicalculatorul **CORAL 4001** este echipat cu două tipuri de memorii operative :

— memorie RAM, pe o singură plachetă, cu module de 16 Kcuvinte (a 18 biți) și 32 Kcuvinte (a 18 biți) ;

— memorie RAM-REPROM de 64 Kocteți, pe o singură plachetă, cu 16 Kcuvinte (a 18 biți) RAM și 16 Kcuvinte (a 16 biți) REPROM, folosită în aplicații industriale (mașini unelte și roboți industriali).

Memoria asigură control de paritate pe octet.

Configurația minimă a unui sistem **CORAL 4001** mai cuprinde o serie de echipamente periferice cu cuploare specifice (pe o singură plachetă) :

— ceas de timp real programabil ;
— consolă operator cu interfață serială asincronă ;
— cititor/perforator de bandă de hîrtie ;
— subsistem disc flexibil simplă densitate ;
— cititor cartele ;
— imprimantă ;
— subsistem disc cartridge de 2,5 Mo ;
— interfață serie asincronă (dual㠋) ;
— multiplexor asincron cu 8 căi ;
— interfață paralelă ;
— interfață sincronă.

Fig. 1.6 reprezintă schema-bloc simplificată a minicalculatorului **CORAL 4001.**

1.3.2. Minicalculatorul CORAL 4001A

Minicalculatorul **CORAL 4001A** are aceleași caracteristici funcționale ca și **CORAL 4001**, fiind o variantă constructivă a acestuia, îmbunătățită din punct de vedere tehnologic, la o structură hardware mai simplă și mai economică.

Caracteristicile îmbunătățite ale minicalculatorului **CORAL 4001A** se referă la :

— utilizarea intensivă a circuitelor de tip LSI, FPLA și PROM.

— utilizarea microprocesoarelor bit-slice de tipul AM 2901 și 8X300 (pentru controloarele de dispozitive periferice) ;

— microprogramare pe orizontală, cu lungimea cuvîntului de 48 biți ; setul complet de instrucțiuni (incluzînd setul **EIS** și **FIS** — pentru lucrul în virgulă mobilă scurtă) este realizat în 512 cuvinte.

Plachetele principale ce fac parte din configurația **CORAL 4001A** sînt :

- **CP01C** — Unitatea Centrală cu emulator de consolă și controlor de **BUS**, pe 6 straturi ;
- **Panou** fără registre de adrese, date și comutatori.

Sertarul minicalculatorului poate conține 6 sau 12 plăci.

Caracteristicile generale ale Unității Centrale sînt similare cu cele ale **CORAL 4001**. Memoriile operative utilizate sînt similare cu cele ale **CORAL 4001**.

O serie de cuploare de echipamente periferice au fost reproiectate pentru asigurarea unei mai mari densități la același număr de plăci :

- **MLC40** — memorie 64 Kocteți, interfață asincronă și ceas de timp real ;
- **LPC40** — interfață cititor cartele/imprimantă ;
- **FD41** — subsistem disc flexibil dublă densitate ;
- **QB40** — interfață de conversie a semnalelor între **BUS**-ul **CORAL 4001A** și **Q-BUS**-ul minicalculatoarelor **PDP-11**, pentru asigurarea unei compatibilități între cele două tipuri de procesoare și dispozitive periferice.

Schema-bloc simplificată a minicalculatorului **CORAL 4001A** este identică cu cea a **CORAL 4001** (fig. 1.6).

1.3.3. Minicalculatorul CORAL 4011

Tehnologic, minicalculatorul **CORAL 4011** are o structură compactă, pe două plachete (de format mare, una cu 6 și alta cu 4 straturi), utilizînd intensiv circuite integrate de tip LSI.

Blocul aritmetico-logic este similar cu cel de la **CORAL 4001**, utilizînd microprocesoare bit-slice, de tipul AM2901, microprogramare pe orizontală (cu lungimea cuvîntului de 64 biți) ; setul standard de instrucțiuni include înmulțirea și împărțirea hardware (**EIS**).

Minicalculatorul **CORAL 4011** este compatibil cu minicalculatorul **PDP-11/34** al firmei **DEC**, SUA.

Magistrala de comunicație, **BUS**, este similară cu cea a minicalculatorului **CORAL 4001**, avînd însă 22 linii de adresă (adresarea pe **BUS** este limitată însă în cazul **CORAL 4011** la 256 Kocteți). Viteza maximă de transfer a informațiilor este tot de 10 Mo/secundă.

În scopul creșterii posibilităților de adresare la memorie, Unitatea centrală include o unitate de relocare și protecție a memoriei ce asigură :

- posibilitatea de adresare pe **BUS** a 256 Kocteți ;
- spațiu de adresare virtual de 16 biți pentru un spațiu fizic de 22 biți ;
- două moduri de operare : *Sistem (Kernel)* și *Utilizator (User)* ;
- două stive program (cîte una pentru fiecare mod) ;
- 16 pagini de relocare (cîte 8 pentru fiecare mod).

Unitatea Centrală operează cu cuvinte de 16 biți, iar adresa generată pe **BUS** este de 22 biți, diferența de 6 biți fiind adăugată de dispozitivul de relocare și protecție a memoriei.

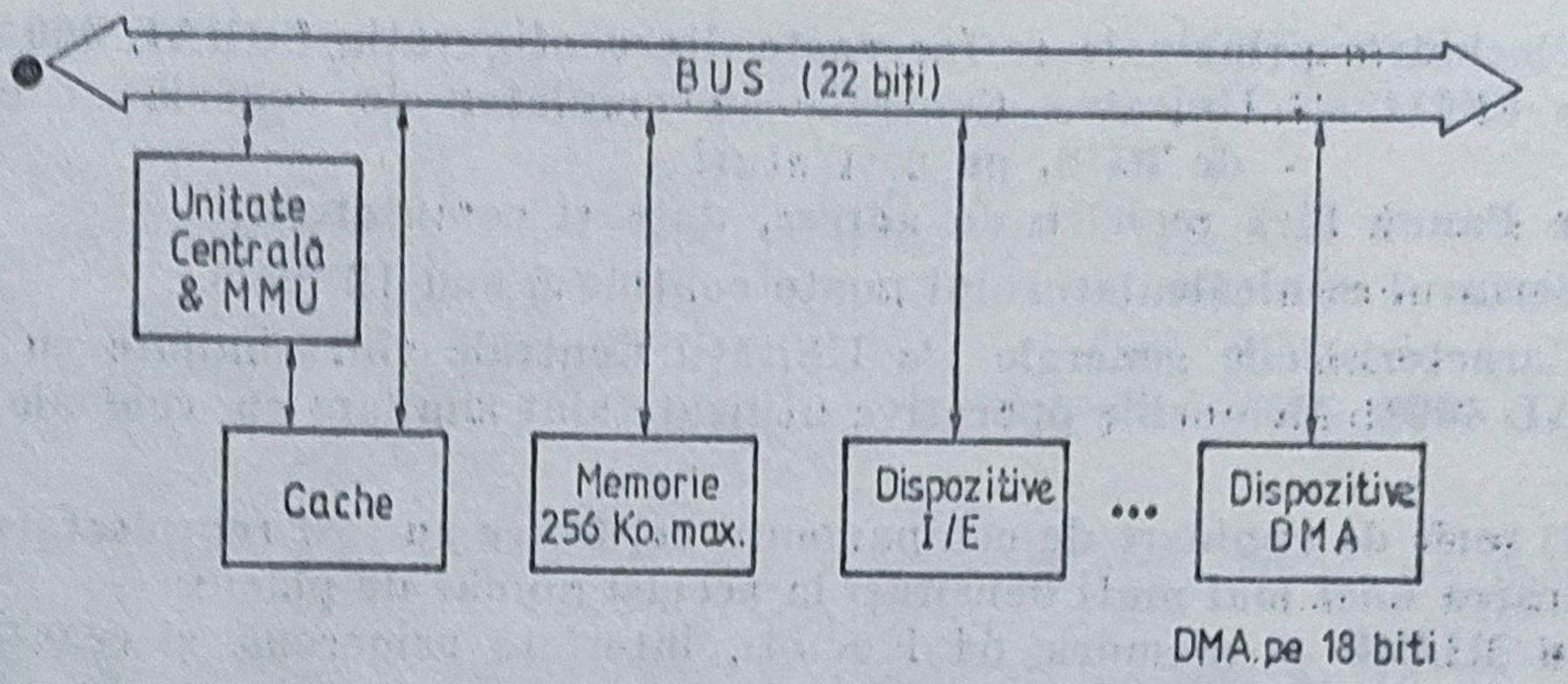


Fig. 1.7. Schema-bloc simplificată a minicalculatorului **CORAL 4011**

În scopul măririi vitezei de execuție a Unității Centrale (ce ajunge la 600 000 operații/secundă), minicalculatorul **CORAL 4011** include o *memorie cache*, ce conține o porțiune anterior selectată a memoriei principale pentru un acces foarte rapid.

Utilizarea memoriei cache este transparentă software, mecanismul de alocare și reactualizare a conținutului memoriei fiind automat.

Memoria cache are o capacitate de 2 Kocteți. Structura cuvîntului din memoria cache, cu lungimea de 27 de biți, conține biți de date, biți de recunoaștere, bitul de validare și 3 biți de paritate.

Plachetele principale ce fac parte din configurația **CORAL 4011** sînt următoarele :

- **CP11** — Unitatea Centrală pe 2 plachete ;
- **BC40** — Interfață panou ;
- **Panou** cu registru de comutatori.

Sertarul minicalculatorului poate conține 12, 18 sau 24 plachete.

Memoria operativă a minicalculatorului **CORAL 4011** este de tip RAM, pe o singură plachetă, cu două variante de implantare :

— 16 Kcuvinte (a 18 biți) ;
— 32 Kcuvinte (a 18 biți).

Configurația de echipamente periferice, cu cuploare specifice (în general pe o singură plachetă), conține echipamente similare cu cele prezentate la **CORAL 4001**. În plus, minicalculatorul **CORAL 4011** include noi echipamente, cum sînt :

- **MD40** — subsistem disc de masă de 50 Mo ;
- **MT40** — subsistem bandă magnetică dublă densitate 800/1600 bpi.

Fig. 1.7 reprezintă schema-bloc simplificată a minicalculatorului **CORAL 4011**.

1.3.4. Minicalculatorul CORAL 4011A

Caracteristicile îmbunătățite ale minicalculatorului **CORAL 4011A** sînt similare cu cele ale **CORAL 4001A** :

— utilizarea intensivă a circuitelor de tip LSI, FPLA, PROM și a microprocesoarelor AM 2901 și 8X300 ;

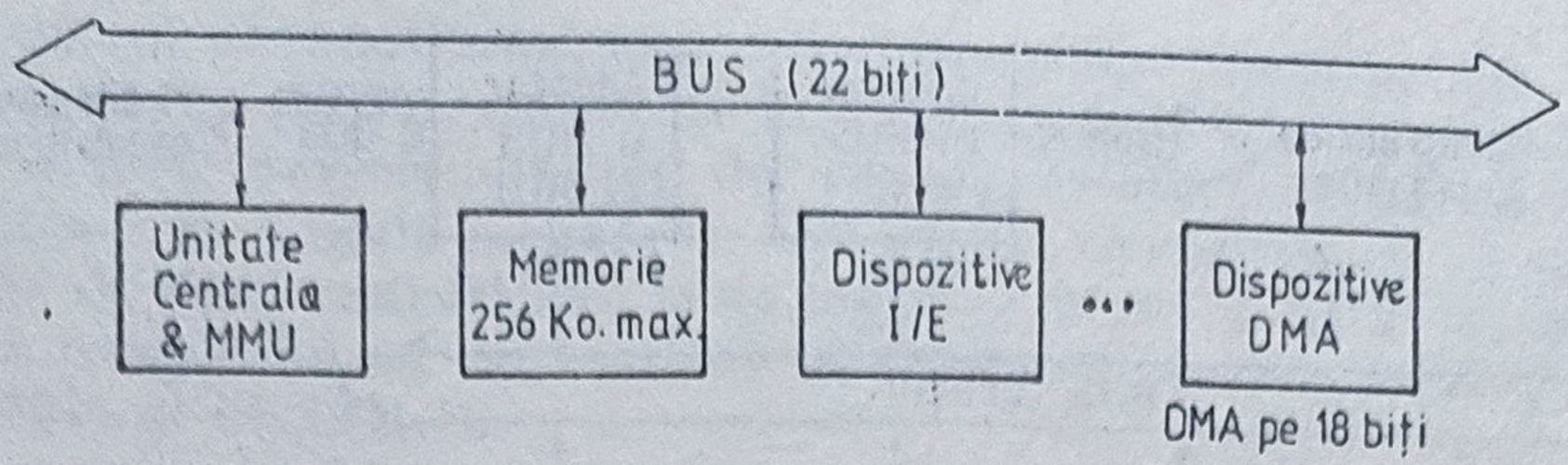


Fig. 1.8. Schema-bloc simplificată a minicalculatorului **CORAL 4011A**

— microprogramare pe orizontală cu cuvînt de microinstrucțiune de 48 biți ;

— setul standard include setul restrîns de instrucțiuni de lucru în virgulă mobilă scurtă **(FIS)** ;

— fără memorie cache ;

— utilizează memorii RAM de 256 Kocteți pe o singură plachetă.

Planchetele principale ce fac parte din configurația **CORAL 4011A** sînt :

- **CP11A** — Unitate Centrală cu emulator de consolă și controler de **BUS**, pe două plachete (una cu 6 straturi, alta cu 4 straturi) ;
- **Panou** fără registre de date, adrese și comutatori.

Fig. 1.8 reprezintă schema-bloc simplificată a minicalculatorului **CORAL 4011A.**

1.3.5. Minicalculatorul CORAL 4030

Tehnologic și constructiv, minicalculatorul **CORAL 4030** este derivat din minicalculatorul **CORAL 4011A.**

Principala caracteristică a minicalculatorului **CORAL 4030** o constituie extinderea adresării la memorie și pe **BUS**, de la 256 Ko la 4 Mo. Arhitectural, el devine astfel comparabil cu **PDP-11/24** (orientat **Q-BUS**) și **PDP-11/44** (orientat **UNIBUS**), păstrînd caracteristici de implementare originale.

Setul standard de instrucțiuni include setul restrîns de instrucțiuni de lucru în virgulă mobilă scurtă **(FIS).**

Utilizează memorii RAM de 256 Ko **(MM256A)** sau 1024 Ko **(MM1024)**, pe o singură plachetă.

Un efort important a fost depus pentru reproiectarea cuploarelor dispozitivelor periferice cu acces **DMA.** Spre deosebire de minicalculatoarele din familia **INDEPENDENT, I-102F** cu extensie de memorie, **I-102/4M, I-106** și **CESAR-16,** unde există un bloc funcțional special de mapare a magistralei de dispozitive de I/E **(BUS-MAP),** ce asigură conversia adresei de 18 biți în adresă de 22 biți pe magistrala de memorie, arhitectura minicalculatorului **CORAL-4030** permite maparea directă a dispozitivelor cu acces **DMA** la memorie.

În acest caz, adresa de memorie pe 22 de biți se transmite dispozitivelor cu acces **DMA** în două registre specifice, din setul de registre de stare și control atașate dispozitivului, plasate în pagina externă :

— un registru normal pe 16 biți, de adresare la memorie ;

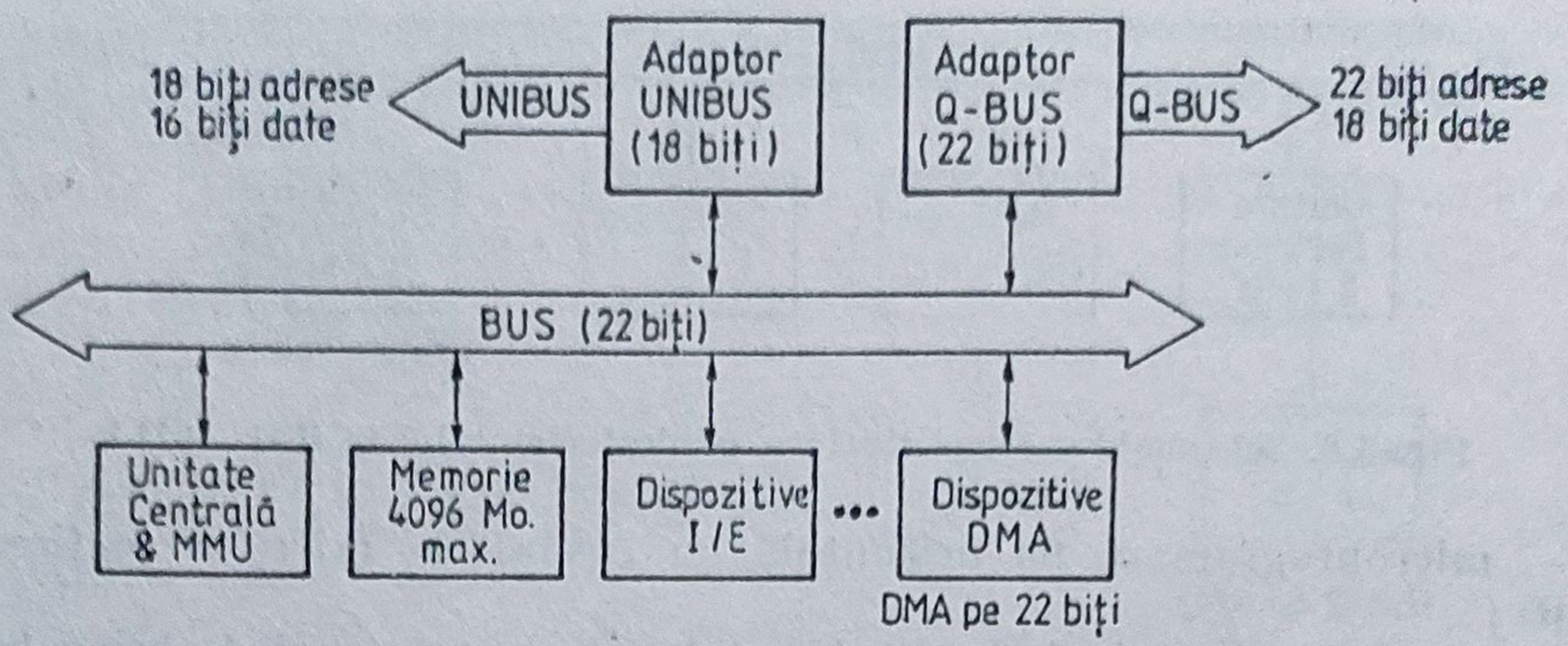


Fig. 1.9. Schema-bloc simplificată a minicalculatorului CORAL 4030

— un registru special, de extensie, pe 16 biți (din care se utilizează numai primii 6), de adresare extinsă la memorie.

Cuplorul unui asemenea dispozitiv cu acces **DMA** se poate folosi, fără modificări, și pe minicalculatoarele **CORAL 4011** și **CORAL 4011A**, caz în care nu se folosește registrul de extensie ; trebuie, însă precizați (într-un registru existent de control și stare) biții 17 și 18 de adresare la memorie (pentru a putea accesa 256 Ko).

Configurația de echipamente periferice cuprinde toată gama existentă pe celelalte modele de minicalculatoare **CORAL**. În plus, minicalculatorul **CORAL 4030** include noi echipamente, cum sînt :

- **MX100/200** — subsistem disc de masă de 100 sau 200 Mo ;
- **MPX40** — procesor de comunicații tip KMC.

Pentru cuplarea unor echipamente compatibile cu cele existente la seria **PDP-11**, există adaptoare specifice de **UNIBUS** și **Q-BUS**.

Fig. 1.9 reprezintă schema-bloc simplificată a minicalculatorului **CORAL 4030**.

1.3.6. Minicalculatorul CORAL 4021 (DP21)

Tehnologic, și constructiv, minicalculatorul **CORAL 4021 (DP21)** este derivat din minicalculatorul **CORAL 4030**, bazîndu-se pe o nouă Unitate Centrală, de mare performanță, **CP21**.

Procesorul **CP21** decodifică și execută atît setul standard de instrucțiuni, cît și setul de instrucțiuni în virgulă mobilă (**FPP**). Procesorul conține o unitate de relocare și protecție a memoriei ce permite extinderea adresării pînă la 4 Mo de memorie (adrese de 22 de biți).

Tehnologic, procesorul **CP21** este construit în jurul microprocesoarelor bit-slice AM 2901B și al memoriei RAM de tipul 82S09. Memoria de microprograme, cu capacitatea de 2 Kcuvinte a 64 biți, este realizată cu 8 circuite integrate de tip 82S191. Se utilizează microprogramarea pe orizontală, ceea ce permite realizarea mai multor funcții într-o singură microstare, reducînd numărul de pași necesari unei instrucțiuni.

În vederea obținerii unui raport preț-performanță competitiv, *procesorul de virgulă mobilă* (**FPP**) este inclus în procesorul de bază, folosind aceleași resurse hardware. Instrucțiunile de virgulă mobilă pot adresa 6 registre acumulatoare de virgulă mobilă, ce sînt văzute ca registre de 32 sau 64 biți, în funcție de tipul instrucțiunii și de registrul de stare. Setul de instrucțiuni **FPP** este compatibil cu cel existent la minicalculatoarele **I-102F, I-102/4M, I-106 I-1016** și **CESAR-16.**

În vederea obținerii unei viteze de execuție mai ridicate, procesorul **CP21** utilizează *suprapunerea de faze în execuția instrucțiunilor* (folosind un registru pipeline de 64 biți), permițînd aducerea instrucțiunii următoare din memorie, simultan cu execuția instrucțiunii curente (overlap). Viteza de execuție este de cca 800 000 instrucțiuni registru cu registru pe secundă.

Implementarea unui *emulator de consolă microprogramat* (asemănător cu cel de la **PDP-11/44**) permite utilizatorului operarea minicalculatorului fără a avea acces la panoul sistemului. Emulatorul poate examina registrele generale ale Unității Centrale, registrele acumulatoare de virgulă mobilă, etc., fiind de un real folos în depanarea hardware și software.

Arhitectural, minicalculatorul **CORAL 4021** este comparabil cu **PDP-11/24** și **PDP-11/44**, avînd caracteristici de implementare originale.

O caracteristică interesantă a procesorului **CP21** o constituie posibilitatea conectării pe un **BUS** bidirecțional separat (cu 21 linii de adresă), a unei memorii cuprinse între 256 Ko și 2 Mo. De regulă, se folosește un plan de memorie de 1 Mo (**MS1024**), special proiectat pentru acest sistem, cu acces dual. Restul de memorie, pînă la 4 Mo, va fi conectat pe BUS-ul general al sistemului (putîndu-se folosi module de 256 Ko sau 1 Mo — **MM1024**).

Procesorul **CP21** este realizat pe 2 plachete mari de circuit imprimat, cu 4 straturi, avînd masa și alimentarea pe straturile interne, ceea ce îi mărește imunitatea la zgomot.

În configurarea sistemelor **CORAL 4021** se pot folosi toate interfețele și cuploarele de dispozitive periferice existente ale familiei de minicalculatoare **CORAL** (în special cele ale **CORAL 4030** pentru adresarea extinsă a memoriei). Minicalculatorul **CORAL 4021** include și noi echipamente, cum sînt :

- **WD40** — subsistem disc Winchester cu capacitatea de 20 sau 40 Mo ;
- **IMC40** — interfață microprogramată de comunicații de mare viteză (tip KMC-DMC) ;
- **AD40** — interfață analog digitală, etc.

iar o parte din cuploarele existente au fost reproiectate :

- **MT41** — interfață de bandă magnetică 800/1600 bpi ;
- **AI44** — 4 interfețe asincrone seriale (cu control de modem) ;
- **LPI44** — 4 interfețe de imprimantă ;
- **BP09** — 4 interfețe asincrone seriale (fără control de modem), interfață de imprimantă serială, ceas de timp real.

Fig. 1.10 reprezintă schema-bloc simplificată a minicalculatorului **CORAL 4021.**

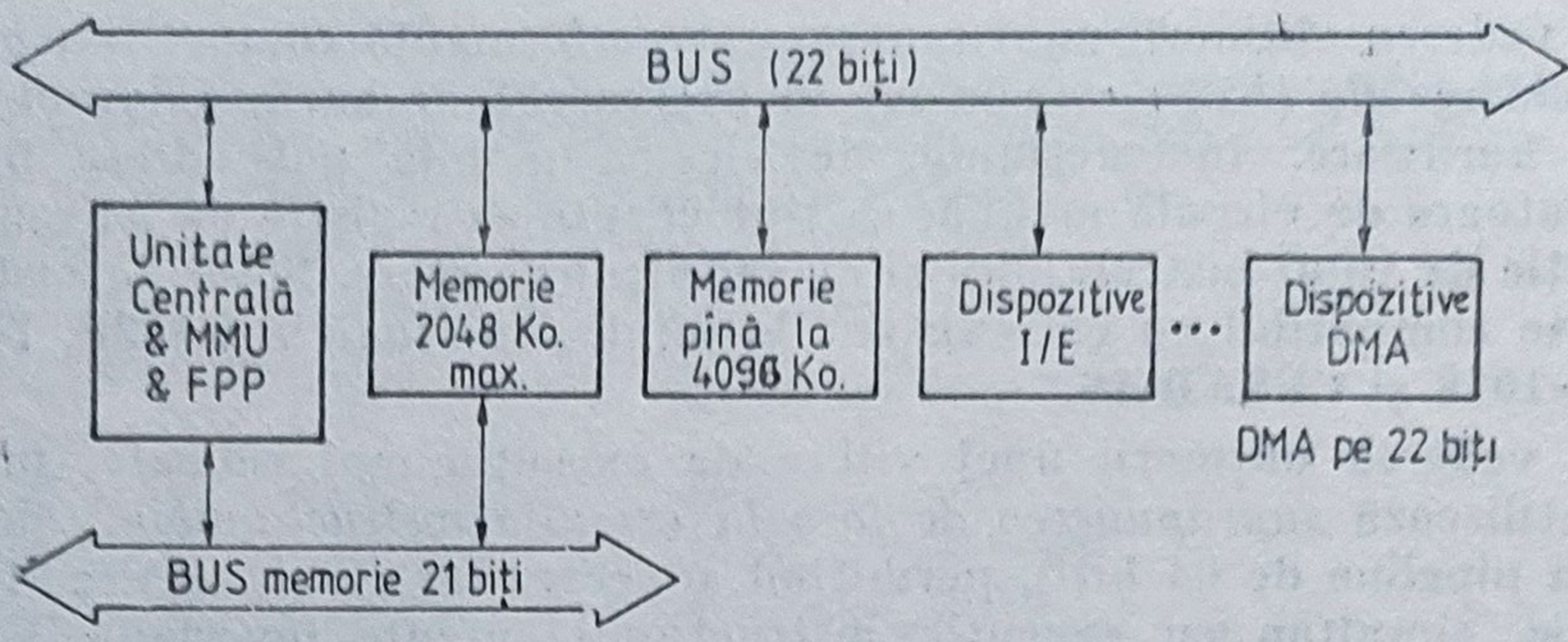


Fig. 1.10. Schema-bloc simplificată a minicalculatorului **CORAL 4021**

1.3.7. Minicalculatorul CORAL 4015 (DP15)

Minicalculatorul **CORAL 4015 (DP15)** are aceleași caracteristici ca și **CORAL 4021**, constituid o variantă constructivă și tehnologică îmbunătățită a acestuia.

Unitatea Centrală a minicalculatorului **CORAL 4015 (CP15)** este realizată pe o singură plachetă de circuit imprimat, cu 4 straturi. Pe **BUS**-ul separat de adresare la memorie se pote conecta minimum 256 Ko (**MS256**) și maximum 1024 Ko (**MS1024**).

Viteza de execuție este de cca. 600 000 operații pe secundă (durata unei operații de adunare fiind de 1,8 μsec). Arhitectural, minicalculatorul **CORAL 4015** este compatibil cu **PDP-11/23 PLUS**, avînd caracteristici de implementare originale.

Minicalculatorul **CORAL 4015** este livrat într-o configurație extrem de compactă, de tip piedestal („calorifer"), cu șasiu de 6 plachete, cuprinzînd :

- **CP15** — procesor pe 2 plachete ;
- **MS1024** — memorie 1024 Ko pe o singură plachetă ;
- **FD41** — subsistem disc floppy dublă densitate, pe o singură plachetă ;
- **WD40** — subsistem disc Winchester cu capacitatea de 40 Mo, pe o singură plachetă ;
- **BP09** — ceas de timp real, interfață de imprimantă (serială), 4 interfețe asincrone seriale (fără control de modem).

Schema-bloc simplificată a minicalculatorului CORAL 4015 este asemănătoare cu cea a lui **CORAL 4021** (fig. 1.10).

1.4. Minicalculatoare românești pentru medii severe și speciale

În paralel cu dezvoltarea familiilor de minicalculatoare **INDEPENDENT** și **CORAL**, destinate rezolvării unor probleme de interes general, universal, în cadrul Institutului de Cercetări pentru Tehnică de Calcul **(ITC)**, București, s-au proiectat și realizat în regim de microproducție două minicalculatoare, **CBEX** și **CESAR-16**, destinate unor aplicații ce presupun condiții mecano-climatice deosebit de severe, atît în regim staționar cît și mobil/ambarcat.

1.4.1. Minicalculatorul CBEX

În realizarea minicalculatorului **CBEX** au fost investigate și folosite tehnologii constructive specifice realizării unor echipamente de tehnică de calcul ultrafiabilă ; împachetarea ansamblului electronic în incinte etanșe, realizarea unui suport tehnologic (circuite imprimante) specific, etc.

Tehnologia constructivă este bazată pe standardul **ARINC-ATR** (*Aeronautical Radio, Inc.*), cu dimensiuni și module standard. Minicalculatorul **CBEX** este împachetat într-un modul **ATR** (496, 887 * 257, 175 * 193, 075), ce include 20 de plăci imprimate și sursa de alimentare, din care 9 plăci sînt atribuite modulelor procesorului și memoriei iar restul de 11 reprezintă o gamă variată de periferice standard sau specializate.

Unitatea Centrală a minicalculatorului **CBEX** execută setul de bază și setul extins (**EIS**) de instrucțiuni al modelului de referință **PDP-11/34**, al firmei **DEC**, SUA. Unitatea Centrală are o structură microprogramată, lungimea cuvîntului de microprogram este de 70 de biți, iar emulatorul de instrucțiuni ocupă o memorie de microprograme de 0,5 kcuvinte. Printr-o proiectare atentă a cuvîntului de microprogram, s-a reușit emularea a cca 50% din setul de instrucțiuni într-un singur pas de microprogram, ceea ce duce la obținerea unei viteze de calcul de aproximativ 650 000 operații pe secundă. Tehnologic, în realizarea Unității Centrale, s-au folosit microprocesoare din seria **AM 2900** și circuite de memorie cu capacitatea de 16 kbiți fiecare (memorie maximă 256 kocteți).

Pentru testarea și punerea la punct a tuturor modulelor procesorului se folosește un sistem modular de microprograme. Minicalculatorul **CBEX** asigură o compatibilitate totală de execuție cu setul de instrucțiuni al famililor **INDEPENDENT** și **CORAL**.

Dispozitivul principal de introducere a datelor și programelor este discul flexibil simplă densitate.

Modalitățile speciale de răcire ale ansamblului **CBEX** (folosirea soluției „perete rece" și sistem de radiație la nivelul fiecărui circuit integrat), permit funcționarea corectă a minicalculatorului într-o gamă largă de temperatură, $-40°C \div +60°C$, și în condiții de presiune scăzută (cca 20 mbar), ducînd la utilizarea sa în medii marine și aeriene.

Schema bloc simplificată a minicalculatorului **CBEX** este asemănătoare cu cea a minicalculatorului **CORAL 4011** (fig. 1.7).

1.4.2. Minicalculatorul CESAR-16

Minicalculatorul **CESAR-16** (Calculator Electronic Special pentru Aplicații în timp Real), este destinat unor aplicații în timp real și este realizat într-o tehnologie care asigură o fiabilitate ridicată, corespunzînd standardului militar **CMEA 067-81** pentru grupa **N7-U-11-A** și caracteristicilor din categoria de exploatare **N3**, conform **STAS 6535-83** și **6692-82** (medii staționare și mobile).

CESAR-16 este un minicalculator puternic, robust și fiabil, realizat pe plăci standard **EUROCARD** (**DEE** 232 * 220 mm), cu două sau patru stra-

turi, folosind microprocesoare din seria **AMD 2900** (sau compatibile, **KP 1804**), circuite integrate **MSI** și circuite de memorie de 64 kbiți (**KP565PU5**).

Minicalculatorul **CESAR-16** poate conține mai multe module-sertar pentru plachete **DEE**, fiecare modul avînd **20** de plachete, sistem de ventilație propriu și panou frontal, echipat cu mufe de tip **KPT41**, prin care se asigură legătura dintre Unitatea Centrală și echipamentele periferice și de proces.

Minicalculatorul **CESAR-16** este microprogramat și emulează setul de instrucțiuni al minicalculatorului **PDP-11/34** al firmei **DEC**, SUA. Elementele componente ale minicalculatorului sînt organizate în jurul unei magistrale interne de comunicație, de tip **UNIBUS**, ce conține 16 linii de date și 18 de adresă (adresare directă a maximum 256 kocteți). În scopul adresării unei memorii de pînă la 4 Mocteți, unitatea de relocare și protecție a memoriei include un dispozitiv de translatare (mapare) a adreselor (**BUS MAP**), ce asigură și accesul dispozitivelor **DMA** la memoria extinsă.

În scopul măririi vitezei de execuție a Unității Centrale (ce ajunge la cca 500 000 operații/secundă), minicalculatorul **CESAR-16** include o *memorie cache*, cu o capacitate de 2 kocteți, transparentă software.

Unitatea Centrală decodifică și execută atît setul standard de instrucțiuni al familiei **PDP-11** (inclusiv setul extins — **EIS**) cît și setul de instrucțiuni în virgulă mobilă (**FPP**), precizie simplă (16 biți) și extinsă (32 și 64 biți). Setul de instrucțiuni **FPP** este compatibil cu cel existent la minicalculatoarele **I-102F, I-102/4M, I-106, I-1016** și **CORAL 4021**.

Minicalculatorul **CESAR-16** se livrează în mai multe configurații, ce conțin următoarele tipuri de Unități Centrale :

— **2900 A** — ce poate adresa pînă la 64 ko memorie (ca și **CORAL 4001**) ;

— **2900 B** — ce poate adresa pînă la 256 ko memorie (ca și **CORAL 4011**) ;

— **2900 E** — ce poate adresa pînă la 4 Mo memorie (ca și **I-102/4M**).

Memoriile operative sînt de două tipuri (64 ko și 256 ko), cu control de paritate și recuperare din eroare (**ECC**).

Unitățile de repertoriu comercial (**URC**) acoperă o gamă largă de cuploare pentru diverse tipuri de echipamente periferice și pentru cuplarea la proces :

— cuploare mici : consolă, ceas de timp real de linie, ceas de timp real programabil, cititor de bandă perforată ;

— cuploare pentru imprimante paralele (lentă și rapidă) ;

— cuplor pentru disc flexibil, dublă densitate ;

— cuplor pentru disc încasetat (2,5 Mo și 5 Mo) ;

— cuplor pentru disc Winchester (cu capacitate de 10—40 Mo) ;

— cuplor pentru bandă magnetică de 1600 bpi ;

— cuplor și memorie de masă semiconductoare (**MEGASTORE**) (cu capacitate de 2 Mo, 4 Mo și 8 Mo) ;

— cuplor de interfață paralelă (cu sau fără **DMA**) ;

— cuplor multiplexor asincron cu 8 căi, tip **DZ11** ;

— cuplor multiplexor sincron cu 4 căi, tip **DUP11** ;

— procesor frontal pentru comunicații, tip **KMC11** ;

— cuploare de intrări/ieșiri numerice ;

— cuploare de intrări/ieşiri analogice ;
— dulap industrial pentru module **DEE** etc.

Schema bloc simplificată a minicalculatorului **CESAR-16** este asemănătoare cu cea a minicalculatorului **CORAL 4021** (fig. 1.10), cu un singur **BUS** şi translatare de tip **BUS MAP**.

1.5. Magistrale de comunicaţie (BUS-uri)

1.5.1. Generalităţi

Magistrala de comunicaţie (sau **BUS**-ul) reprezintă o componentă cheie a arhitecturii minicalculatoarelor româneşti, constituind singurul sistem de comunicare între diversele module ale sistemului de calcul. Acest sistem utilizează un set unic de semnale pentru comunicarea procesorului cu echipamentele periferice şi memoria. Unitatea centrală, memoria şi toate dispozitivele periferice partajează aceeaşi cale ; aceasta înseamnă că registrele de stare şi control ale dispozitivelor periferice pot fi adresate ca orice locaţie de memorie şi că transferul de date între două dispozitive periferice sau între memorie şi un dispozitiv periferic se face fără a mai trece prin Unitatea centrală.

Nu există nici o instrucţiune specială de intrare/ieşire. Toate instrucţiunile minicalculatoarelor româneşti sînt disponibile pentru efectuarea de intrări/ieşiri.

Transmiterea de date şi adrese pe **BUS** se face bidirecţional şi asincron. Controlul **BUS**-ului se face pe baza unei structuri de priorităţi hardware ataşate fiecărui dispozitiv periferic.

Arhitectura **BUS**-ului diferă pentru fiecare familie de minicalculatoare în parte.

Familia de minicalculatoare **INDEPENDENT (I-100, I-102F** şi **I-102/4M)** utilizează 2 **BUS**-uri :

— **INTERBUS** — ce asigură legătura între Unitatea Centrală şi echipamentele periferice ale sistemului ;
— **MEMOBUS** — ce asigură accesul la memorie a unităţii centrale şi o cale rapidă de transfer a dispozitivelor **DMA**.

Minicalculatoarele **I-106** şi **I-1016**, spre deosebire de celelalte modele din familie **(I-100** şi **I102F)** utilizează un singur **BUS : SBUS** — la care sînt conectate toate dispozitivele periferice şi memoria.

Familia de minicalculatoare **CORAL (4001(A), 4011(A), 4030, 4021, 4015)** precum şi **CESAR-16** utilizează un singur **BUS : BUS** — ce constituie sistemul de comunicare între toate tipurile de module din configuraţia de calcul.

Fig. 1.11 prezintă schema generală a magistralelor minicalculatoarelor **I-100, I-102F** şi **I-102/4M.**

Fig. 1.12 reprezintă conectarea echipamentelor periferice şi memoriei pentru minicalculatorul **I-106** şi **I-1016**, iar fig. 1.13 situaţia similară de pe minicalculatoarele **CORAL** şi **CESAR-16.**

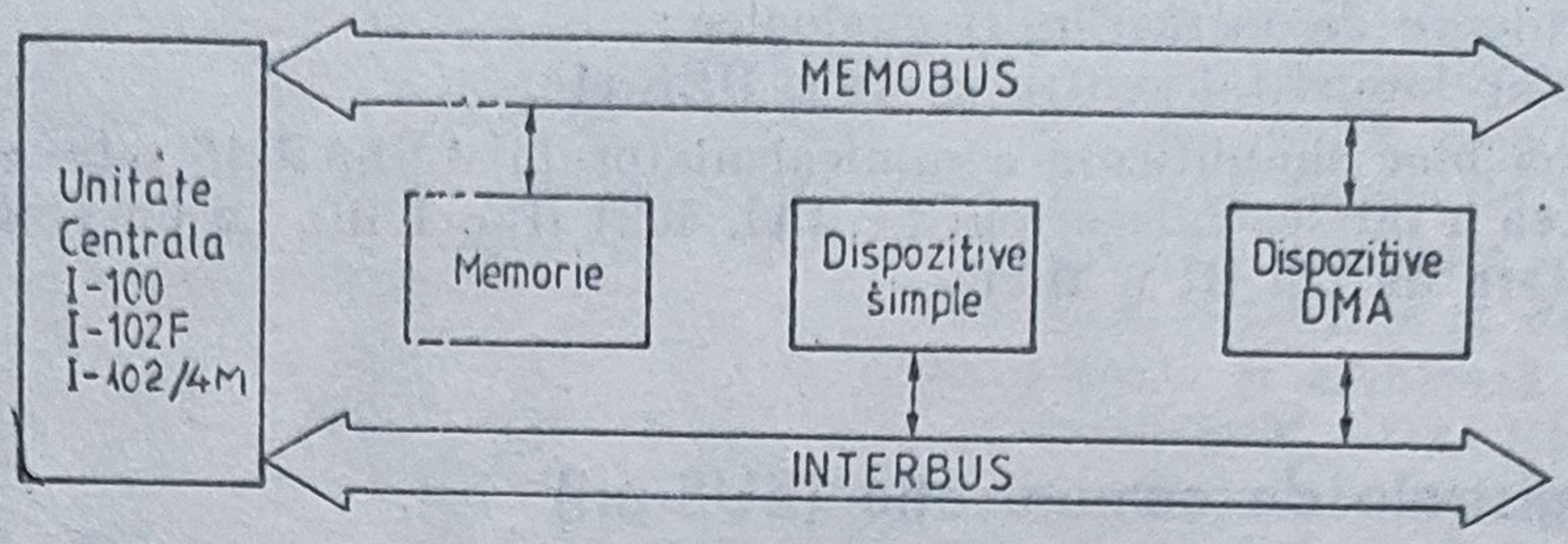


Fig. 1.11. Magistralele minicalculatoarelor I-100, I-102F și I-102/4M

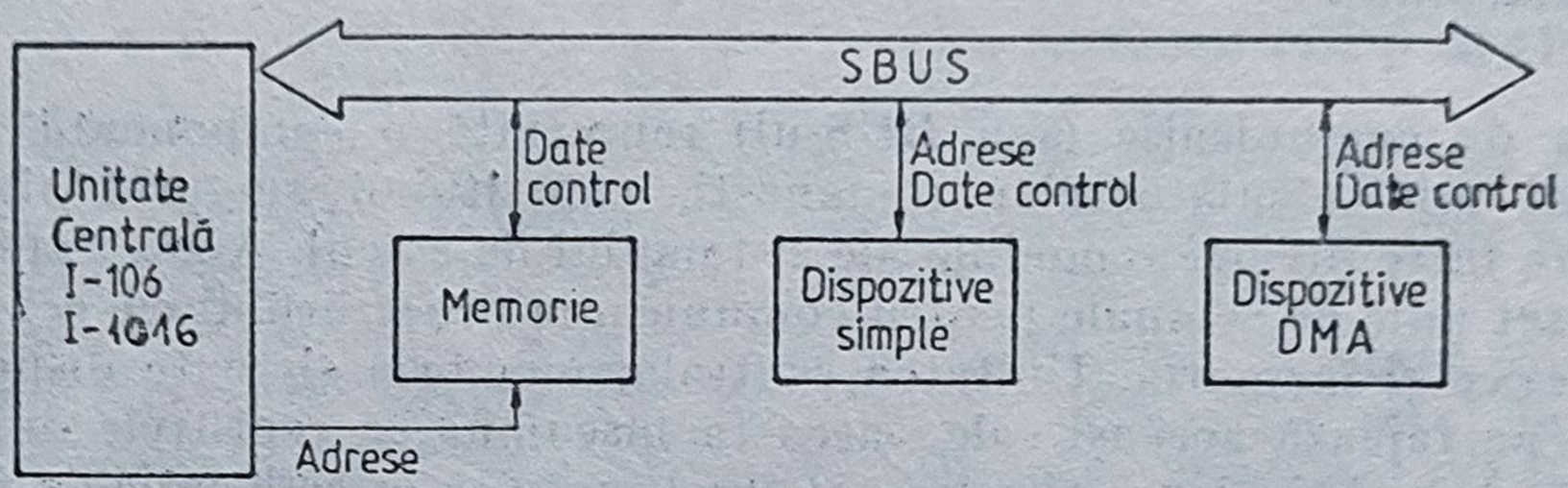


Fig. 1.12. Conectarea echipamentelor periferice și a memoriei la minicalculatoarele I-106 și I-1016

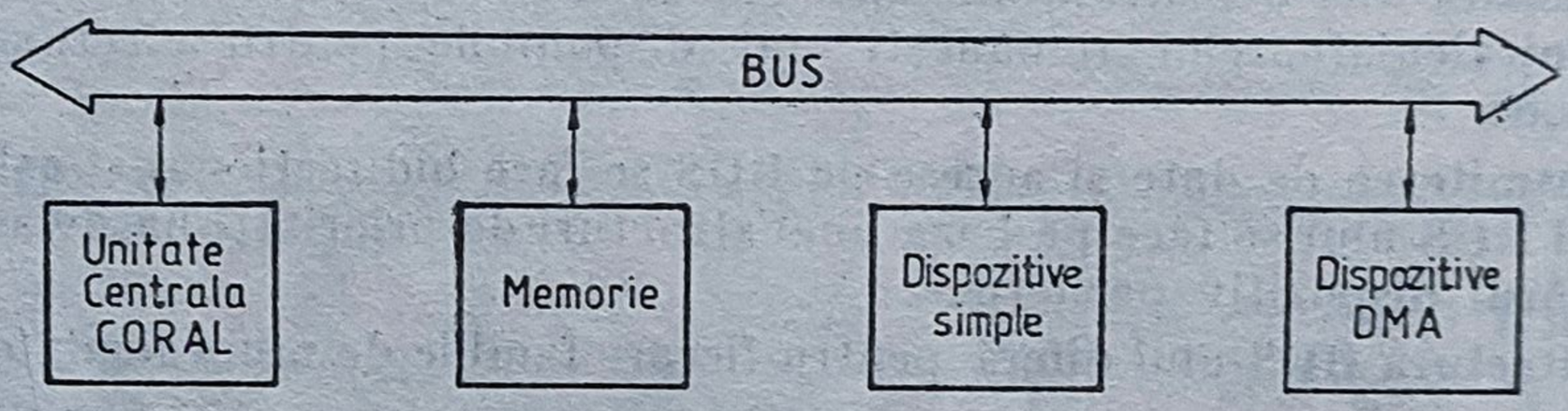


Fig. 1.13. Conectarea echipamentelor periferice și a memoriei la minicalculatoarele CORAL 4001/4001A, 4011/4011A, 4030, 4021, 4015 și CESAR-16

1.5.2. Principii de organizare a magistralelor

Magistralele de Comunicație prezente în familiile de minicalculatoare **INDEPENDENT (INTERBUS, MEMOBUS** sau **SBUS), CORAL (BUS),** respectă aceleași principii generale de organizare și funcționare.

Magistralele vehiculează semnale reprezentînd comenzi, stări, adrese și date. Majoritatea semnalelor folosesc linii bidirecționale, la care sînt conectate toate dispozitivele care trebuie să transmită sau să recepționeze semnale pe aceste linii. Altă categorie de semnale folosesc linii unidirecționale.

Fiecare magistrală de comunicație are însă o serie de caracteristici particulare, specifice arhitecturii minicalculatoarelor românești :

— **INTERBUS** — conține 72 de linii, din care 18 pentru adresă și 16 pentru date.

— **MEMOBUS** — conține 49 de linii, din care 22 (la **I-102F** cu extensie memorie, **I-102/4M**), 21 (la **I-100**) sau 18 (la **I-102F** fără extensie de memorie) sînt pentru adresă și 16 pentru date.

— **SBUS** — conține 18 (**I-106**) sau 22 (**I-1016**) linii pentru adresă și 16 pentru date

— **BUS** — conține 16 (la **CORAL 4001(A)**), 16 (la **CORAL 4011(A)**, **CESAR-16**) sau 22 (la **CORAL 4030, 4021, 4015**) linii pentru adresă și 16 pentru date.

Liniile de adresă din structura magistralei, ce definesc posibilitățile de adresare de memorie, sînt :

— 18 = **INDEPENDENT I-100, I-102, I-106; I-1016, CESAR-16** adresare directă a 32 Kc, cu relocare 124 Kc memorie ;

— 22 = **CORAL 4001(A)**, adresare directă a 32 Kc memorie ;

— 22 = **CORAL 4011(A)**, adresare directă a 32 Kc, cu relocare 124 Kc memorie ;

— 22 = **CORAL 4030, 4021, 4015** adresare directă a 32 Kc, cu relocare 2048 Kc memorie.

Memoria are asociate adrese situate în primii 248 Ko din totalul de 256 Ko ai spațiului de adresare (minicalculatoarelor **I-100, I-102F** fără extensie de memorie, **CORAL 4011(A)**) sau în primii 3860 Ko din totalul de 4096 Ko (minicalculatoarele **I-102F** cu extensie de memorie, **I-102/4M, I-106, CESAR-16** sau în primii 4088 Ko din totalul de 4096 Ko (**CORAL 4030, 4021, 4015, 1-1016**).

La minicalculatoarele **I-100** și **I-102F** (fără extensie de memorie), atunci cînd adresarea se face către Unitatea Centrală, accesul este direcționat către **MEMOBUS**, pentru adrese cuprinse în primii 248 Ko ai spațiului de adresare, sau **INTERBUS** (pentru adrese cuprinse în ultimii 8 Kocteți — corespunzătoare registrelor dispozitivelor periferice).

La minicalculatorul **I-106**, spre deosebire de dispozitivele periferice, memoria nu are conectate la **SBUS** decît liniile de control și date, avînd însă linii de adresă separate (deoarece memoria necesită 22 linii de adresă, corespunzătoare unui spațiu total, de adresare de 4 Mo, iar **SBUS**-ul dispune doar de 18 linii de adresă, corespunzătoare unui spațiu de 256 Ko). Conversia între adresele pe 18 biți de pe **SBUS** și adresele de 22 biți ale memoriei este realizată de către un dispozitiv special de translatare, numit **BUS MAP**.

O situație similară există și la minicalculatoarele **I-102F** cu extensie de memorie, **I-102/4M** și **CESAR-16**. Conversia între adresele pe 18 biți de pe **INTERBUS** și adresele de 22 biți ale memoriei este realizată de asemenea de către dispozitivul special de translatare, **BUS MAP**.

La minicalculatoarele **CORAL** și **I-1016**, accesul la memorie și dispozitivele periferice se face, în general, pe aceeași magistrală unică de comunicație de tip **BUS**. Dacă unitatea de relocare și protecție a memoriei (**MMU**) este inactivă, toate referințele la ultimii 8 Ko, în spațiul de 64 Ko, sînt convertite în adrese fizice de 22 biți, avînd biții 16—21 puși pe 1. Dacă unitatea de relocare este activă, se păstrează convenția anterioară pentru spațiul de 256 Ko (**CORAL 4011(A)**) sau 4096 Ko (**CORAL 4030, 4021, 4015, I-1016**).

Minicalculatoarele **CORAL 4021** și **4015**, pentru rapidizarea accesului la memorie, cuprind și un **BUS** specific de acces direct al unității centrale la o memorie de 1 Mo. Dacă accesul se face la o memorie mai mare de 1 Mo, se utilizează magistrala de comunicație principală (**BUS**), folosită și pentru accesarea dispozitivelor periferice.

Comunicarea între două dispozitive pe magistrală este de tipul „*master/slave*". Unul din dispozitive are controlul magistralei *(master)* și o comandă în timpul comunicării cu un alt dispozitiv *(slave)*. Un exemplu tipic este transferul de date între Unitatea Centrală (ca master) și memorie (ca slave).

Relația „*master-slave*" este dinamică. La un moment dat pe magistrală poate să fie master Unitatea Centrală, iar în momentul următor un dispozitiv periferic tip **DMA** (disc sau bandă magnetică, de exemplu) care să transfere date în/din memorie, considerată „*slave*".

Comunicarea între dispozitivele conectate la o magistrală este intercondiționată, fiecare semnal de comandă emis de dispozitivul „*master*" trebuind să fie confirmat de răspunsul dispozitivului „*slave*".

Comunicarea este independentă de lungimea magistralei sau de timpul de răspuns al dispozitivelor, acest gen de dialog conferind magistralei caracterul său asincron.

1.5.3. **Conectarea dispozitivelor periferice la magistrale**

Dispozitivele de intrare/ieșire se conectează la magistralele sistemului prin cuploarele lor. Fiecare cuplor conține un set de registre adresabile de pe magistrală, fiecare registru având asociată o adresă la care răspunde. Programele adresează aceste registre ca pe niște locații obișnuite de memorie, putîndu-le utiliza ca operanzi sursă sau destinație în orice instrucțiune aritmetică sau logică.

Registrele dispozitivelor periferice au adresele situate în ultimii 8 Kocteți ai spațiului de adrese fizice ale minicalculatoarelor (pagina externă sau pagina de intrare/ieșire).

Semnificația și funcțiile acestor registre diferă de la un dispozitiv periferic la altul. Există registre de control și stare, registre de date, etc.

O operație de intrare/ieșire cu un echipament periferic implică transferul unui număr de caractere între memoria calculatorului și echipamentul periferic.

O serie de dispozitive (cum ar fi terminalele, imprimanta, etc.) necesită intervenția Unității Centrale pentru fiecare caracter transferat. Tipărirea unui mesaj pe un terminal implică scrierea de către Unitatea Centrală a fiecărui caracter în registrul de date al cuplorului și așteptarea tipăririi acestuia.

Alte dispozitive (cum ar fi discul sau banda magnetică) necesită intervenția Unității Centrale doar pentru inițierea operației de intrare/ieșire. Dispozitivului periferic i se furnizează parametrii transferului: adresa de memorie, lungimea zonei de transferat, adresa disc, codul funcției, etc. După

inițierea operației, dispozitivul periferic execută singur operațiile de transfer între el și memorie. Un astfel de dispozitiv se numește cu acces direct la memorie (**DMA**).

Cîștigarea magistralei se face printr-un proces de arbitrare, în urma cererii emise de dispozitivul periferic.

1.5.4. Sistemul de întreruperi

Pentru ca un program să realizeze o operație de intrare/ieșire, el trebuie informat asupra momentului în care echipamentul periferic a terminat de executat operația de intrare/ieșire (integral, în cazul perifericelor de tip **DMA**, sau pe caracter, în cazul celorlalte periferice). Atenționarea programului este posibilă prin funcționarea dispozitivelor în regim de întreruperi. Fiecare dispozitiv are în registrul de control și stare un bit care, atunci cînd este poziționat de către program, permite generarea unei întreruperi la sfîrșitul operației de intrare/ieșire.

Acordarea dreptului de întrerupere pentru un dispozitiv se face tot printr-un proces de arbitrare, pe baza existenței unei structuri de priorități.

Sistemul de întreruperi al minicalculatoarelor românești este organizat pe mai multe nivele de prioritate, la fiecare putîndu-se atașa mai multe dispozitive periferice.

Există 8 nivele de întreruperi standard (**BR**) numerotate de la 7 la 0, nivelul 7 fiind cel mai prioritar. Ultimele 4 nivele inferioare (3 la 0) nu sînt utilizate, putînd corespunde în sistemul de operare unor nivele de priorități software.

Atașarea unui dispozitiv la un nivel specific determină prioritatea hardware a acestuia. Prioritatea mai multor dispozitive atașate pe același nivel este dată de poziția acestora ; dispozitivele mai apropiate de UC au o prioritate mai mare.

Dispozitivele cu acces direct la memorie (**DMA**), cum ar fi discurile, benzile magnetice, etc., își transferă datele fără intervenția Unității Centrale, în paralel cu execuția unui program în Unitatea Centrală. O situație similară apare și în cazul unui transfer între două dispozitive periferice.

Aceste transferuri, care nu necesită intervenția Unității Centrale (**NPR**), sînt prioritare transferurilor pe nivelele de priorități 7—0.

Sistemul de întreruperi și priorități al minicalculatoarelor românești este prezentat în fig. 1.14.

La întreruperile de pe nivelele 7—4 corespund, în mod unic, vectori de întreruperi în memoria internă, ceea ce duce la eliminarea polling-ului pentru determinarea sursei unei întreruperi (în programele sistem sau de aplicație).

În momentul apariției unei întreruperi, contorul de instrucțiuni (adrese) curent (**PC**) și starea program (**PS**) sînt salvate în stiva curentă. Unitatea Centrală încarcă din vectorul de întreruperi specific sursei de întrerupere un nou **PS** și **PC**. Tratarea întreruperii se face la nivelul de prioritate indicat de noul **PS**.

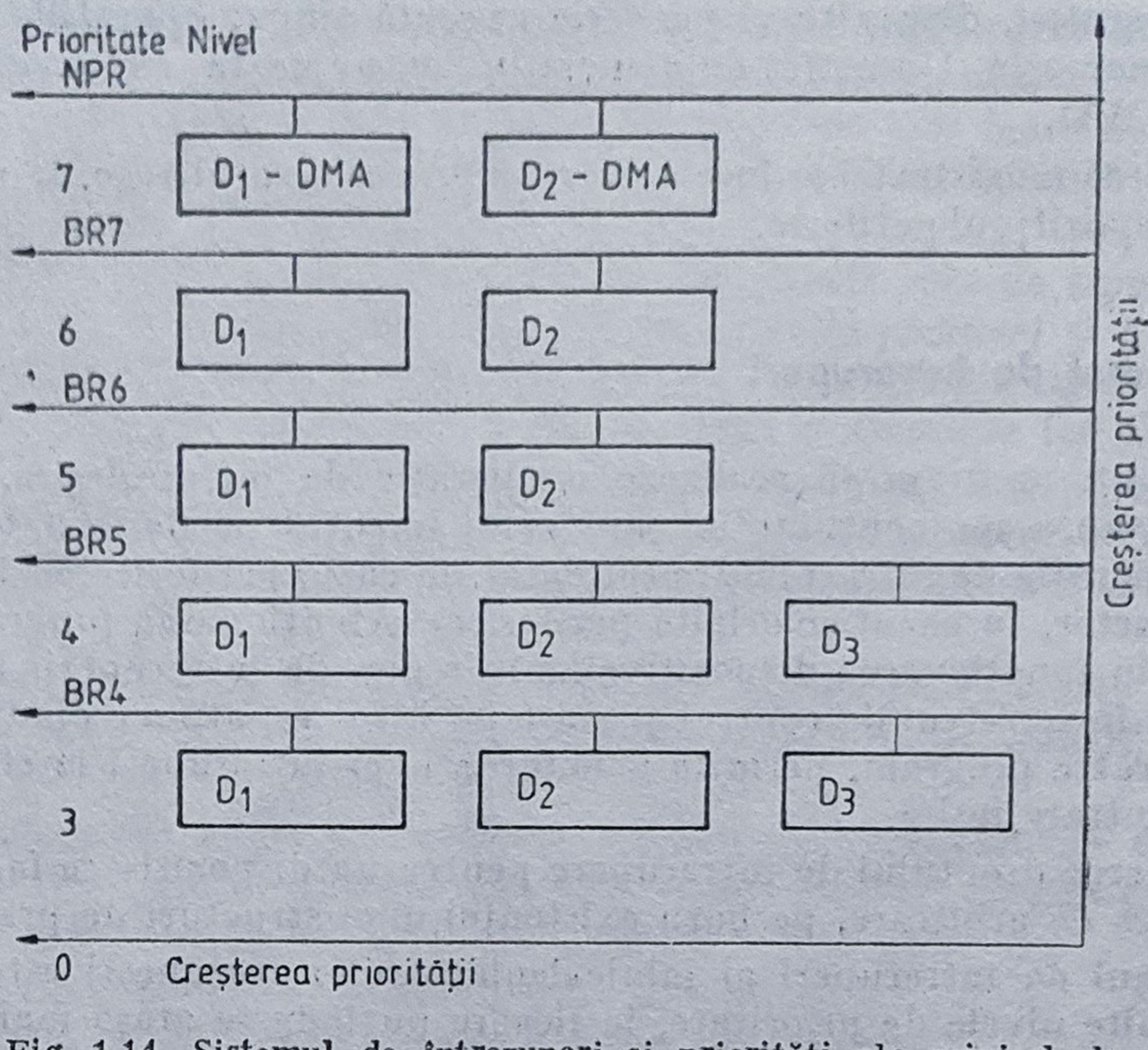


Fig. 1.14. Sistemul de întreruperi și priorități ale minicalculatoarelor românești

1.6. Unitatea Centrală

Unitatea Centrală a minicalculatoarelor românești, conectată la **BUS/INTERBUS/SBUS** ca un subsistem, îndeplinește următoarele funcțiuni :

— supraveghează operațiile de transfer pe magistrale și arbitrează nivelele de prioritate hardware ;

— execută operații artimetice și logice ;

— decodifică și execută instrucțiuni.

1.6.1. Registre generale

Unitatea Centrală conține 9 registre generale (la minicalculatoarele **I-100, I-102F, I-102/4M, CORAL, CESAR-16**) sau 10 registre generale (la minicalculatoarele **I-106, I-1016**), utilizabile în diferite scopuri : acumulatori, registre index, registre cu autoincrementare sau autodecrementare, indicatori de stivă pentru memorarea temporară a datelor.

Operațiile aritmetice pot fi efectuate între un registru general și altul, între o locație de memorie sau registru de dispozitiv și altul de acest tip sau între o locație de memorie sau registru de dispozitiv și un registru general.

Toate cele 9 (respectiv 10) registre sînt accesibile programelor :

— registrele **R0, R1, R2, R3, R4, R5** au o semnificaţie obişnuită ;

— registrul **R6** este utilizat în mod standard, de către Unitatea Centrală, ca indicator de stivă (**SP**). Pe minicalculatoarele **I-100, I-102F, I-102/4M, CORAL, CESAR-16** există doi indicatori de stivă diferiţi, corespunzători celor două moduri de lucru ale Unităţii Centrale : *Sistem (Kernel)* şi *Utilizator (User)*. Pe minicalculatoarele **I-106** şi **I-1016** există un indicator de stivă suplimentar, corespunzător celui de-al treilea mod de lucru al Unităţii Centrale : *Supervizor (Superviser)* ;

— registrul **R7** este utilizat drept contor de instrucţiuni (adrese) al programului (**PC**) şi conţine adresa instrucţiunii ce urmează a fi executată.

1.6.2. **Setul de instrucţiuni**

Spre deosebire de minicalculatoarele convenţionale cu lungime de cuvînt pe 16 biţi, cu trei clase de instrucţiuni (referinţă la memorie, instrucţiuni de control şi instrucţiuni de I/E), toate operaţiile din minicalculatoarele româneşti sînt efectuate cu un singur set de instrucţiuni.

Lungimea instrucţiunii este variabilă, de la unul la trei cuvinte de 16 biţi, în funcţie de modul de adresare utilizat.

Instrucţiunile pot fi fără operanzi, cu un singur operand sau doi operanzi, adresarea făcîndu-se la nivel de octet sau cuvînt.

Formatele generale ale instrucţiunilor minicalculatoarelor româneşti sînt prezentate în tabela A.1 din anexă.

Puterea minicalculatoarelor româneşti este dată, în mare parte, de posibilităţile lor de adresare : adresare indirectă, adresare pe cuvînt sau pe octet şi adresarea stivei.

Un sumar al modurilor de adresare este prezentat în Tabela A.2 din anexă.

Setul standard de instrucţiuni al minicalculatoarelor româneşti este setul de instrucţiuni al modelului de referinţă **PDP-11/34(A)**, prezentat în tabela A.3 din anexă.

Opţional, acest set poate fi extins cu alte două grupe de instrucţiuni :

— de operaţii aritmetice (**EIS**), prezentat în tabela A.4 din anexă ;

— de virgulă flotantă scurtă (**FIS**), prezentat în tabela A.5 din anexă.

Începînd din anul 1981, toate minicalculatoarele româneşti includ grupele de instrucţiuni **EIS** şi **FIS** în setul standard de instrucţiuni.

Minicalculatorul **INDEPENDENT I-102F** emulează setul de instrucţiuni al modelului de referinţă **PDP-11/60** (exceptînd formatul de microinstrucţiune). Extensiile faţă de setul standard de instrucţiuni sînt prezentate în tabela A.7 din anexă. Opţional, acest minicalculator poate include, în locul **FIS**, un set de instrucţiuni în virgulă flotantă lungă (**FPP**), prezentat în tabela A.6 din anexă.

Din anul 1982, grupa de instrucţiuni **FPP** este introdusă în setul standard de instrucţiuni al minicalculatorului **I-102F**.

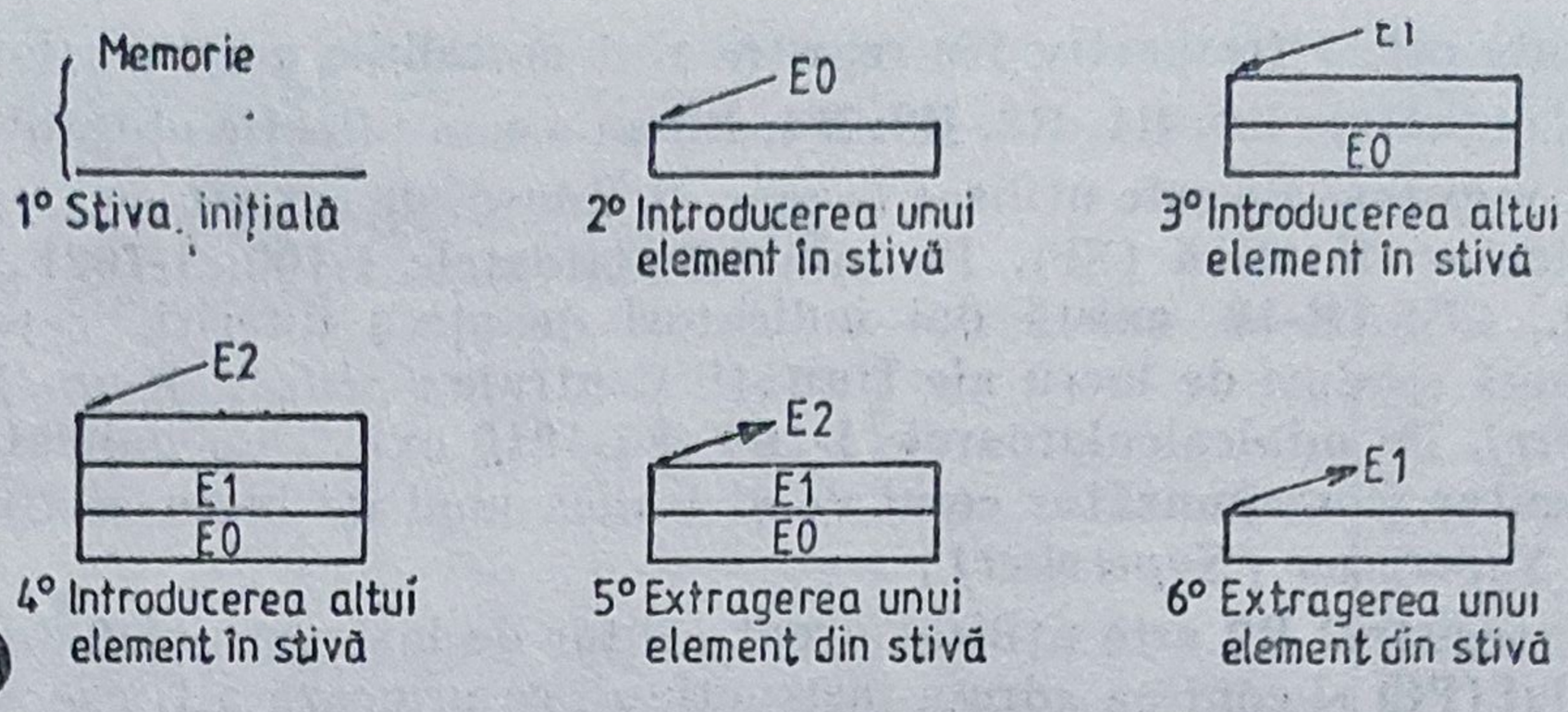


Fig. 1.15. Operațiile de introducere și extragere din stivă

Setul de instrucțiuni în virgulă flotantă lungă (**FPP**) este inclus, în prezent, în setul standard de instrucțiuni al minicalculatoarelor **I-102/4M, I-106, I-1016, CORAL 4021, 4015** și **CESAR-16.**

Minicalculatorul **I-106** poate conține, opțional, un set de instrucțiuni comerciale (**CIS**), prezentat în tabela A.8 din anexă.

1.6.3. Conceptul de stivă

În minicalculatoarele românești, stiva reprezintă o zonă de memorare temporară, ce permite programelor o utilizare eficientă a datelor.

Stiva este o listă **LIFO** (Ultimul introdus — Primul extras), în care se pot introduce sau extrage cuvinte sau octeți prin operații de autoincrementare și autodecrementare.

Unitatea Centrală utilizează structura de stivă pentru salvarea și refacerea contextului unui program (la apariția întreruperilor hardware și software) și în cazul apelurilor de subrutine. Registrul utilizat drept indicator de stivă este, în acest caz, registrul **R6 (SP)**, corespunzător modului curent de lucru.

Figura 1.15 ilustrează operațiile de introducere și extragere din stivă.

1.6.4. Starea program

Cuvîntul de *Stare Program* (**PS**), conține informații asupra stării curente a Unității Centrale :

— prioritatea curentă a unității centrale ;
— modurile de lucru curent și anterior ;
— rezultatul operației anterioare, reflectat în codurile de condiții ;
— un indicator de detectare a execuției unei instrucțiuni ce urmează a fi trasată pentru depanare.

Structura cuvîntului de stare program (**PS**), localizat la adresa (17)777776, este prezentată în figura 1.16.

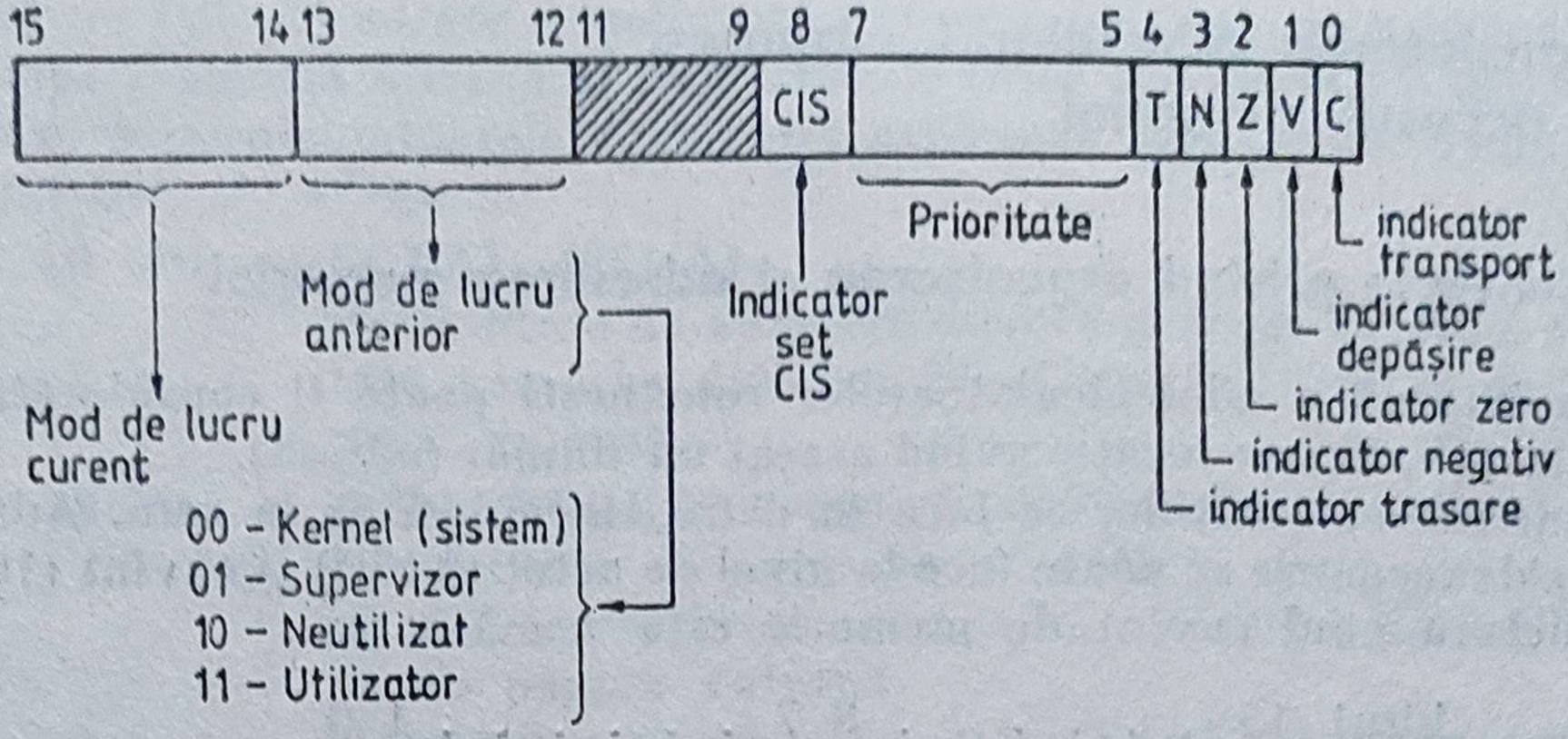


Fig. 1.16. Structura cuvîntului de stare program

1.6.5. Registre specifice unor minicalculatoare

În scopul obținerii unor performanțe de exploatare mai ridicate, Unitățile Centrale ale minicalculatoarelor **I-106** și **I-1016** conțin un set de registre specifice, inexistente la celelalte minicalculatoare românești.

a) **Registrul de eroare al Unității Centrale (CPUERR)**

La apariția unei condiții de eroare, se generează hardware o întrerupere al cărui vector se găsește la adresa 4. În acest caz, registrul **CPUERR** *(CPU Error Register)* conține codul erorii (simplă sau compozită).

Registrul de eroare corespunde adresei 17777766.

b) **Registrul de întreruperi programate (PIRQ)**

În scopul obținerii, de către task-urile privilegiate, a unor întreruperi programate, se poate utiliza registrul de întreruperi programate (**PIRQ** — *Program Interrupt Request)* al Unității Centrale. Cererea de întrerupere se programează poziționînd unul din biții 15÷9 ai registrului (corespunzători unui nivel de prioritate de la 7 la 1). La acceptarea cererii, se forțează o întrerupere programată pe vectorul de la adresa 240.

Registrul de întreruperi programate corespunde adresei 177777772.

c) **Registrul de comutatori (SWR)**

Minicalculatoarele **CORAL 4001(A), 4011(A),** și **I-106** conțin de asemenea un registru de comutatori.

Acest registru (**SWR** — *Switch register)* poate fi încărcat de la cheile panoului de comandă (**CORAL 4001(A), 4011(A)** sau prin consola inteligentă (**I-106**) și este utilizat în semnalizarea unor condiții externe, de către operator, către programul de aplicație (se folosește cu precădere de către programele de test hardware).

Registrul de comutatori corespunde adresei 177777570.

1.7. Translatarea adreselor („maparea") și accesul memoriei

1.7.1. Concepte privind organizarea și adresarea memoriei

Memoria internă a minicalculatoarelor românești poate fi considerată ca o serie de locații, fiecare locație avînd atașat un număr (adresă).

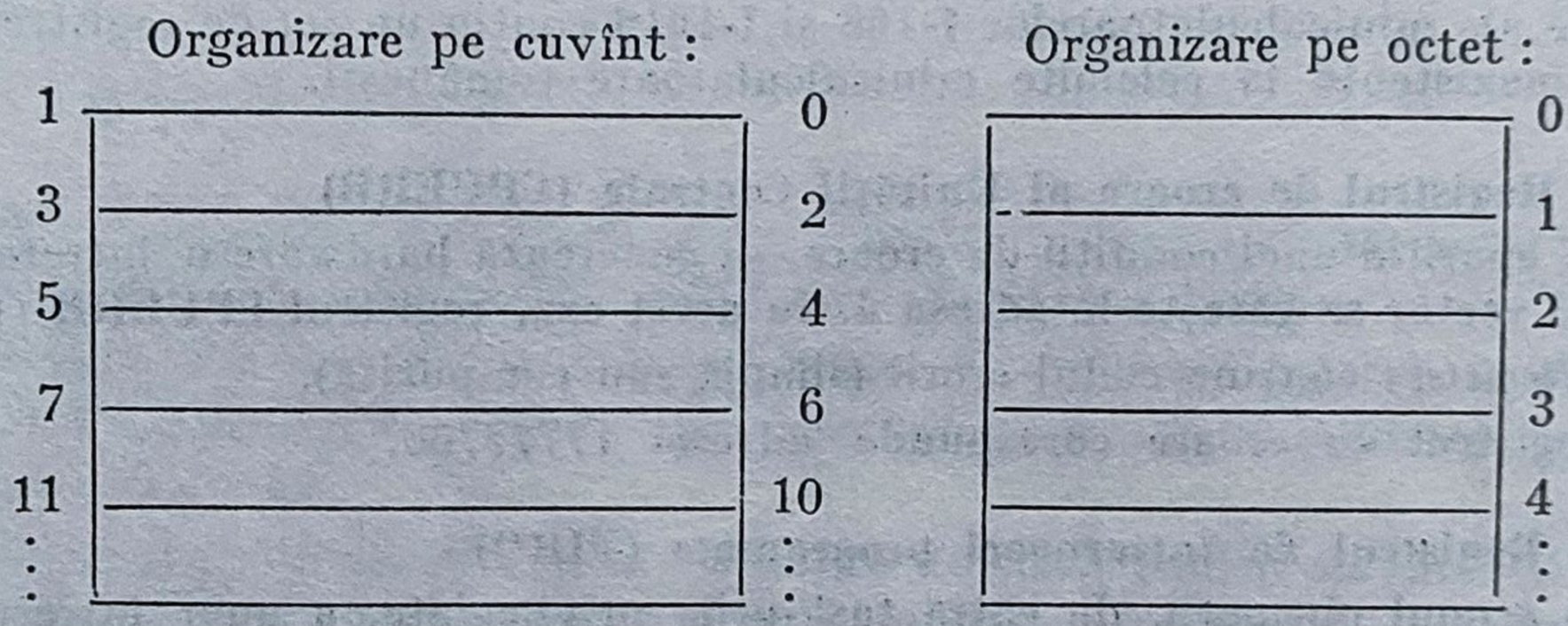
Numerotarea locațiilor se face în octal, începînd de la zero. Adresarea locațiilor de memorie se poate face la nivel de octet (8 biți) și cuvînt (16 biți).

Structura unui cuvînt de memorie este următoarea :

bitul 15 . . . . . 8 7 . . . . . 1 0

Octet superior	Octet inferior

Octeții inferiori (mai puțin semnificativi) sînt memorați la adresc pare, iar octeții superiori (cei mai semnificativi) la adrese impare. În acest sens, memoria arată astfel :



Anumite locații de memorie sînt rezervate de către sistem pentru tratarea întreruperilor hardware și software, drept stive, registre generale și registre de stare și control periferice.

Adresele de la 0 la 2778 sînt întotdeauna rezervate vectorilor de întreruperi software și hardware pentru dispozitivele periferice standard, iar cele de la 300_8 la 777_8 sînt rezervate vectorilor variabili de întreruperi specifici, în general, dispozitivelor de comunicație. Ultimele 4 Kcuvinte de memorie, cuprinse între adresele 770000_8 și 777777_8, sînt întotdeauna rezervate pentru registrele generale și registrele de stare și control ale dispozitivelor periferice.

Această zonă se numește *pagina externă* sau *pagina de intrare/ieșire* a memoriei.

1.7.2. Spațiile de adresare virtual și fizic

Spațiul virtual de adrese reprezintă setul de locații de memorie accesibile într-un program utilizator. Structura pe 16 biți a cuvîntului minicalculatoarelor românești permite o adresare directă a maximum 32 Kcuvinte, între adresele 0 și 177777_8.

Spaţiul fizic de adrese este format din serii contigue de locaţii adresabile ce definesc memoria internă şi regiştrele de stare şi control ale dispozitivelor periferice. Minicalculatoarele româneşti utilizează trei ordine de mărime ale spaţiului fizic de adrese :

— 16 biţi — **CORAL 4001(A)**
Posibilitate de adresare directă şi acces pe BUS a 64 Kocteţi memorie, din care 56 Kocteţi utilizator şi 8 Ko pagina externă.

— 18 biţi — **INDEPENDENT I-100, I-102F, CORAL 4011(A)**
Posibilitate de adresare relocată şi acces direct pe **BUS** a 256 Kocteţi memorie, din care 248 Ko utilizator şi 8 Ko pagina externă.

— 22 biţi — **INDEPENDENT I-102F** extins, **I-102/4M, I-106, CESAR-16.**
Posibilitate de adresare relocată a 4096 Kocteţi memorie, din care 3840 Ko utilizator, 248 Ko pentru maparea dispozitivelor **DMA** şi 8 Ko pagina externă. Întrucît dispozitivele de pe **BUS** sînt direct adresabile printr-o adresă pe 18 biţi, este necesară o mapare separată la memoria fizică.

— **CORAL 4030, 4021, 4015** şi **I-1016.**
Posibilitate de adresare relocată în acces direct pe **BUS** a 4096 Kocteţi memorie, din care 4088 Ko utilizator şi 8 Ko pagina externă.

Extinderea adresării de la 16 biţi la 18 biţi şi 22 biţi se face prin intermediul unui mecanism de relocare a adreselor, implementat hardware în Unitatea Centrală şi cunoscut sub numele de *dispozitiv de relocare şi protecţie a memoriei* (**MMU**).

1.7.3. Dispozitivul de relocare şi protecţie a memoriei (MMU)

Utilizarea *dispozitivului de relocare şi protecţie a memoriei* (**MMU** — *Memory Management Unit*), în minicalculatoarele româneşti, permite :

— extinderea posibilităţilor de adresare a memoriei de la 28 Kc, la 124 Kc. (**I-100/I-102F** fără extensie de memorie, **CORAL 4011(A)**) sau la 2044 Kc (**I-102F** cu extensie de memorie, **I-102/4M, I-106, I-1016, CORAL 4030, 4021, 4015, CESAR-16**) ;

— relocarea unui program în diferite zone ale memoriei ;

— protecţia programelor utilizator între ele.

Caracteristicile generale ale dispozitivului de relocare şi protecţie a memoriei sînt :

— Capacitatea de adresare : 128 Kcuvinte (**I-100/I-102F** fără extensie de memorie, **CORAL 4011(A)**) ;
2048 Kcuvinte (**I-102F** cu extensie de memorie, **I-102/4M, I-106, I-1016, CORAL 4030, 4021, 4015, CESAR-16**) ;

— Spaţiul de adresare :	Virtual : 16 biţi ; Fizic : 18 biţi (I-100/I-102F fără extensie de memorie) ; Fizic : 22 biţi (I-102F cu extensie de memorie, I-102/4M, I-106, I-1016, CORAL 4011(A), 4021, 4015, 4030, CESAR-16) ;
— Spaţii de relocare :	I (Instrucţiuni) ; *D* (Date) (numai pe minicalculatoarele (I-102/4M, I-106 şi I-1016) ;
— Modul de operare :	*Sistem (Kernel) ;* *Supervizor* (numai I-106 şi I-1016) ; *Utilizator ;*
— Stive program : (cîte una pentru fiecare mod)	2 3 (numai I-106 şi I-1016) ;
— Număr de pagini : (cîte 8 pentru fiecare mod)	16 32 (16 pentru spaţiul I, 16 pentru spaţiul D, pe minicalculatoarele I-102F cu extensie de memorie, I-102/4M) ; 48 (24 pentru spaţiul I, 24 pentru spaţiul D, pe minicalculatoarele I-106 şi I-1016) ;
— Lungimea paginii :	32 ÷ 4096 cuvinte ;
— Protecţia memoriei :	Nerezidentă ; Protejată la scriere ; Rezidentă total.

Prezenţa acestui dispozitiv este extrem de necesară şi utilă într-un sistem de multiprogramare sau multi-utilizator, cum este sistemul de operare MIX.

În momentul utilizării acestui dispozitiv, adresa conţinută într-un cuvînt de 16 biţi nu mai este interpretată ca o *adresă fizică* (AF), ci ca o *adresă virtuală* (AV), ce conţine informaţii pentru construirea unei noi adrese fizice pe 18 biţi (I-100/I-102F fără extensie de memorie) sau 22 biţi (I-102F cu extensie de memorie, I-102/4M, I-106, I-1016, CORAL 4011(A), 4030, 4021, 4015, CESAR-16).

Cuvîntul de operare al Unităţii Centrale este de 16 biţi, iar adresa generată de acesta pe BUS este de 18/22 de biţi. Diferenţa de 2/6 biţi este generată de dispozitivul de relocare şi protecţie a memoriei. Trebuie menţionat că alocarea BUS-ului se face în aşa fel încît ultimele 4 Kcuvinte ale spaţiului de adresare sînt rezervate registrelor pe BUS ale periferice lor, procesoarelor şi modulelor de control.

Cînd dispozitivul de relocare şi protecţie a memoriei este inactiv, toate referirile la ultimele 4 Kcuvinte în spaţiul de 16 biţi sînt convertite în adrese fizice de 18/22 de biţi avînd biţii 16—18/21 puşi pe 1 (de ex. referirea la un registru cu adresa 177722 este automat translatată la adresa 777722/17777722). În acest fel procesorul poate adresa direct 28 Kcuvinte de memorie şi 4 Kcuvinte pentru registrele dispozitivelor de pe BUS.

a) Moduri de lucru

Unitățile Centrale ale minicalculatoarelor românești asigură 2 sau 3, (**I-106, I-1016**) moduri de lucru (execuție) a programelor : *Sistem (Kernel)* *Supervizor (Supervisor)* (**I-106, I-1016**) și *Utilizator (User)*, ce îmbunătățesc mecanismul de acces și protecție la memorie și duc la creșterea flexibilității i și performanțelor de exploatare ale unui sistem de operare cu multiprogramare, cum este sistemul **MIX.**

Alegerea unuia sau altuia este determinată de modul curent de lucru descris, de biții 15—14 din Starea Program (**PS**) : (00 — Sistem ; 01 — Supervizor ; 10 — mod ilegal ; 11 — Utilizator).

În *modul Kernel*, programul dispune de un control complet asupra sistemului și poate executa toate instrucțiunile. În acest mod se execută rutinele ce fac parte din Nucleul sistemului de operare, driver-ele, etc. Pagina de intrare/ieșire este de asemenea mapată numai în acest mod pentru a preveni remaparea registrelor de relocare sau modificarea protecției de acces.

În *modul Utilizator*, programul nu are un control complet (nu poate executa toate instrucțiunile — în special **HALT** și **RESET** —, nu poate opri calculatorul, nu poate utiliza spațiul *Kernel*, nu poate modifica registrele de relocare și protecție a memoriei). De obicei, în modul *Utilizator* se execută toate programele utilizator și o bună parte din codul task-urilor privilegiate.

Modul Supervizor are aceleași privilegii ca și modul *Utilizator*, dar se folosește o mapare diferită. În acest mod se execută programe (rutine) partajate între mai mulți utilizatori, protejîndu-se accesul acestora la zona de program(e) comună.

Dispozitivul de relocare și protecție a memoriei utilizează pentru fiecare mod de lucru legal cîte un set de registre. Fiecare set cuprinde cîte 8 perechi de registre : **PAR** — registrul de adresă a paginii și **PDR** — registrul de descriere a paginii. Aceste registre, utilizate întotdeauna împreună, conțin toate informațiile necesare pentru descrierea și relocarea unei pagini.

b) Spații de instrucțiuni și date

Manipularea spațiilor de instrucțiuni (**I**) și date (**D**) constituie o tehnică avansată de programare ce dublează spațiul de adresare virtual al unui program utilizator de la 64 la 128 Kocteți.

Pentru minicalculatoarele românești, **I-102F** cu extensie de memorie, **I-102/4M, I-106** și **I-1016,** dispozitivul de relocare și protecție poate reloca referințele la date și instrucțiuni cu valori separate ale adreselor de bază.

Pentru fiecare mod de lucru (2 sau 3) există cîte 16 perechi de registre de pagină (8 pentru spațiul **I** și 8 pentru spațiul **D**).

În cazul minicalculatoarelor **I-106** și **I-1016,** spațiul de adresare virtual al unui program poate crește de la 128 Ko la 192 Ko (prin folosirea spațiilor de instrucțiuni și date și ale modului de lucru *Supervizor)*.

Programele cu spații **I** și **D** separate trebuie să respecte o serie de reguli pentru a putea funcționa corect :

Spațiul **I** poate conține numai instrucțiuni, operanzi imediați (modul 2 cu **R7**), adrese absolute (modul 3 cu **R7**) și cuvinte index (modurile 6 și 7, orice registru) ;

— pagina ce conține stiva trebuie mapată și în spațiul **I** și în spațiul **D** dacă se intenționează utilizarea instrucțiunii **MARK** (întrucît aceasta este executată în afara stivei) ;

— paginile mapate în spaţiul **D** nu pot conţine parametri de subrutină (procedura de apel standard, folosită pe minicalculatoarele românești, nu poate fi mapată complet în spaţiul **I**). În acest caz, parametrii de apel ai unei subrutine se plasează în stiva curentă.

— Se recomandă maparea zonei de întreruperi hardware și software în ambele spaţii (**I** și **D**).

Restricţiile de utilizare ale modurilor de adresare în spaţiile **I** și **D** sînt prezentate în tabela A.9 din anexă. De remarcat că, pentru ambele spaţii, există cîte un set diferit de registre generale (**R0 ÷ R7**).

Activarea spaţiilor **I** și **D** se face prin intermediul registrului **SR3** conţinut în unitatea de relocare și protecţie a memoriei.

c) **Relocarea adreselor**

Funcţia de bază a dispozitivului **MMU** este de a reloca adresele și proteja memoria, permiţînd totodată extinderea ei peste limita de adresare a cuvîntului de 15 biţi — 32 Kcuvinte.

Cînd dispozitivul de relocare și protecţie a memoriei este activat, adresa de 16 biţi nu este folosită ca adresă fizică, ci ca adresă virtuală, conţinînd informaţiile necesare pentru a constitui adresa fizică de 18/22 de biţi. Informaţia conţinută în adresa virtuală împreună cu informaţia de relocare și de descriere a paginii din registrele **MMU** (**PAR** și **PDR**) este folosită pentru a genera adresa fizică.

Deoarece adresele sînt relocate automat, minicalculatorul poate fi considerat că operează în spaţiul de adresare virtual, acesta fiind divizat în 8 pagini de 4 Kcuvinte fiecare.

Deși spaţiul virtual de adresare al programului este contiguu, fiecare pagină a spaţiului poate fi relocată separat (inclusiv în același spaţiu de memorie fizică).

Maparea aceluiași spaţiu fizic cu registre de relocare diferite (aparţinînd unor programe diferite) constituie o metodă efectivă de partajare/accesare a datelor/informaţiilor între programe cooperante.

Un program este relocat în pagini cu lungimea de la 32 la 4096 cuvinte. Pagina este împărţită în blocuri de 32 cuvinte, putînd avea de la 1 la 128 blocuri. Utilizînd toate cele 8 pagini, lungimea maximă a programului este de 32 768 cuvinte (65 536 octeţi).

În acest scop se folosesc cele două seturi de registre de adresare a paginilor (**PAR/PDR**$_2$).

Registrul de adresă a paginii (**PAR**) este folosit pentru a determina adresa de început a fiecărei pagini relocate în memoria fizică. Fiecare pagină poate fi relocată în memorie la orice adresă multiplu de 32 cuvinte. Pentru paginile mai mici de 4 Kcuvinte numai memoria relocată poate fi accesibilă.

Pentru fiecare mod de lucru (*Sistem, Supervizor, Utilizator*) există un set separat de cîte 8 registre de pagină **PAR/PDR**. De asemenea, pentru fiecare spaţiu de acces (instrucţiuni sau date) există un set separat de cîte 8 registre pe pagină. Pentru programele relocate în spaţiile **I** și **D**, spaţiul virtual al acestora este acoperit de 16 pagini de relocare, lungimea maximă a programului putînd fi de 65 536 cuvinte (131 072 octeţi).

Figura 1.17 prezintă un exemplu de relocare a unui program de 64 Kocteţi într-o memorie fizică de 248 Kocteţi.

Figura 1.8 prezintă mecanismul fizic de selectare a unui set de registre de relocare **PAR/PDR**.

UC

Gama adreselor virtuale
PC:(160000 - 177776)
PC:(140000 - 157776)
PC:(120000 - 137776)
PC:(100000 - 117776)
PC:(060000 - 077776)
PC:(040000 - 057776)
PC:(020000 - 037776)
PC:(000000 - 017776)

MMU

APR	Constantă de relocare
7	1500
6	0200
5	1000
4	0200
3	0600
2	2500
1	3200
0	4000

Memorie fizică

Gama adreselor fizice
400000 - 417776
320000 - 337776
250000 - 267776
150000 - 167776
100000 - 117776
060000 - 077776
020000 - 037776

Fig. 1.17. Exemplu de relocare a unui program în memoria fizică

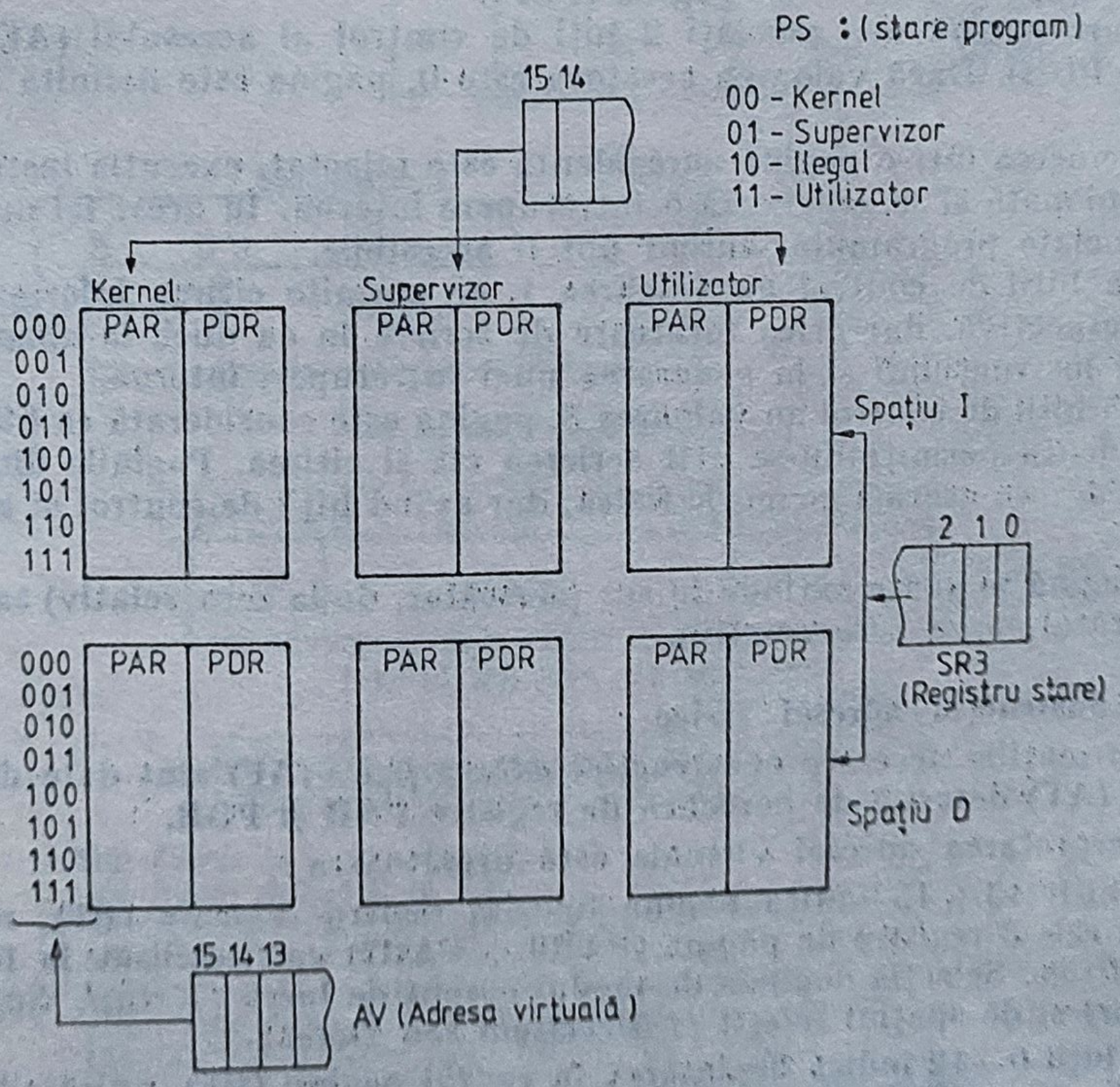


Fig. 1.18. Selectarea setului de registre de relocare PAR/PDR

Registrul de adresă a paginii (**PAR**) conţine 12 biţi (**I-100/I-102F** fără extensie de memorie), **CORAL 4011(A)**) sau 16 biţi (**I-102F** cu extensie de memorie, **I-102/4M, I-106, I-1016, CORAL 4030, 4021, 4015, CESAR-16**) care marchează baza paginii în memoria fizică :

— pentru minicalculatoarele **I-100/I-102F** fără extensie de memorie, **CORAL 4011(A)** :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
[redacted]				17				13				9			6	**PAR**

— pentru minicalculatoarele **I-102F** cu extensie de memorie, **I-102/4M, I-106, I-1016, CORAL 4030, 4021, 4015, CESAR-16 :**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
21				17								9			6	**PAR**

d) **Protecţia memoriei (paginilor)**

Fiecărui registru de adresă a paginii (**PAR**) îi corespunde un registru de descriere a paginii (**PDR**), ce conţine informaţii relative la :

— posibilităţile de acces la pagină (**ACF**) ;

— direcţia de acces la pagină (**ED**) ;

— lungimea de acces în pagină (**PLF**).

Fiecare pagină are asociaţi 2 biţi de control ai accesului (**ACF**) (biţii 1—2 din **PDR**). Dacă valoarea acestora este 0, pagina este definită ca nerezidentă.

Orice acces într-o pagină nerezidentă este rejectat, execuţia instrucţiunii este abandonată şi se generează o întrerupere internă. În acest fel numai paginile asociate programului curent pot fi accesibile.

Dacă biţii de control au valoarea 1, se permite citirea informaţiei din pagina respectivă, dar orice încercare de scriere în ea duce la abandonarea execuţiei instrucţiunii şi la generarea unei întreruperi interne.

Dacă biţii de control au valoarea 3, pagina este considerată ca fiind complet rezidentă permiţîndu-se atît scrierea cît şi citirea. Paginile din fiecare mod pot adresa aceeaşi memorie fizică, dar avînd biţii de control al accesului diferiţi.

O pagină se poate extinde în sus (crescător, de la zero relativ) sau în jos (descrescător, spre zero relativ).

e) **Construcţia adresei fizice**

Informaţiile necesare construcţiei *adresei fizice* (**AF**) sînt date de *adresa virtuală* (**AV**) descrisă de perechea de registre **PAR** şi **PDR**.

Interpretarea adresei virtuale este următoarea :

— biţii 13 ÷ 15 indică pagina folosită pentru relocare (**RP**), respectiv care din cele 8 registre de pagină (**PAR0** ÷ **PAR7**) va fi utilizat la formarea adresei fizice. Selecţia depinde de modul curent de lucru (*Kernel, Supervizor, Utilizator*) şi de spaţiul referit (*Instrucţiuni* sau *Date*) ;

— biţii 0—12 indică deplasarea în cadrul paginii (**DF**), adică adresa relativă la începutul paginii.

DF este divizată în două cîmpuri :

— biții 6—12 reprezintă numărul de bloc (0—127) în cadrul paginii curente **(NBL)** ;

— biții 0—5 reprezintă deplasarea (0—63 octeți) în cadrul blocului **(DIB).**

Formarea adresei fizice din adresa virtuală este prezentată în figura 1.19.

f) **Registre de stare și recuperare din eroare**

Întreruperile și derutările generate de dispozitivul de relocare și protecție a memoriei sînt localizate în spațiul *Kernel*, avînd vectorul de întrerupere la adresa 250_8. Registrele de stare **SR0, SR1** și **SR3** sînt folosite pentru a determina cauza întreruperii și refacerea programului.

Registrul de stare 0 (SR0) — adresa 17777572

Registrul **SR0** conține indicatorii de eroare, de activare a dispozitivului de relocare și protecție a memoriei, numărul de pagină a cărei referire a dus la eroare, modul de lucru în care a apărut eroarea, precum și alți indicatori de stare.

Bitul 0 semnifică activarea (setat) sau inhibarea (șters) dispozitivului de relocare și protecție a memoriei.

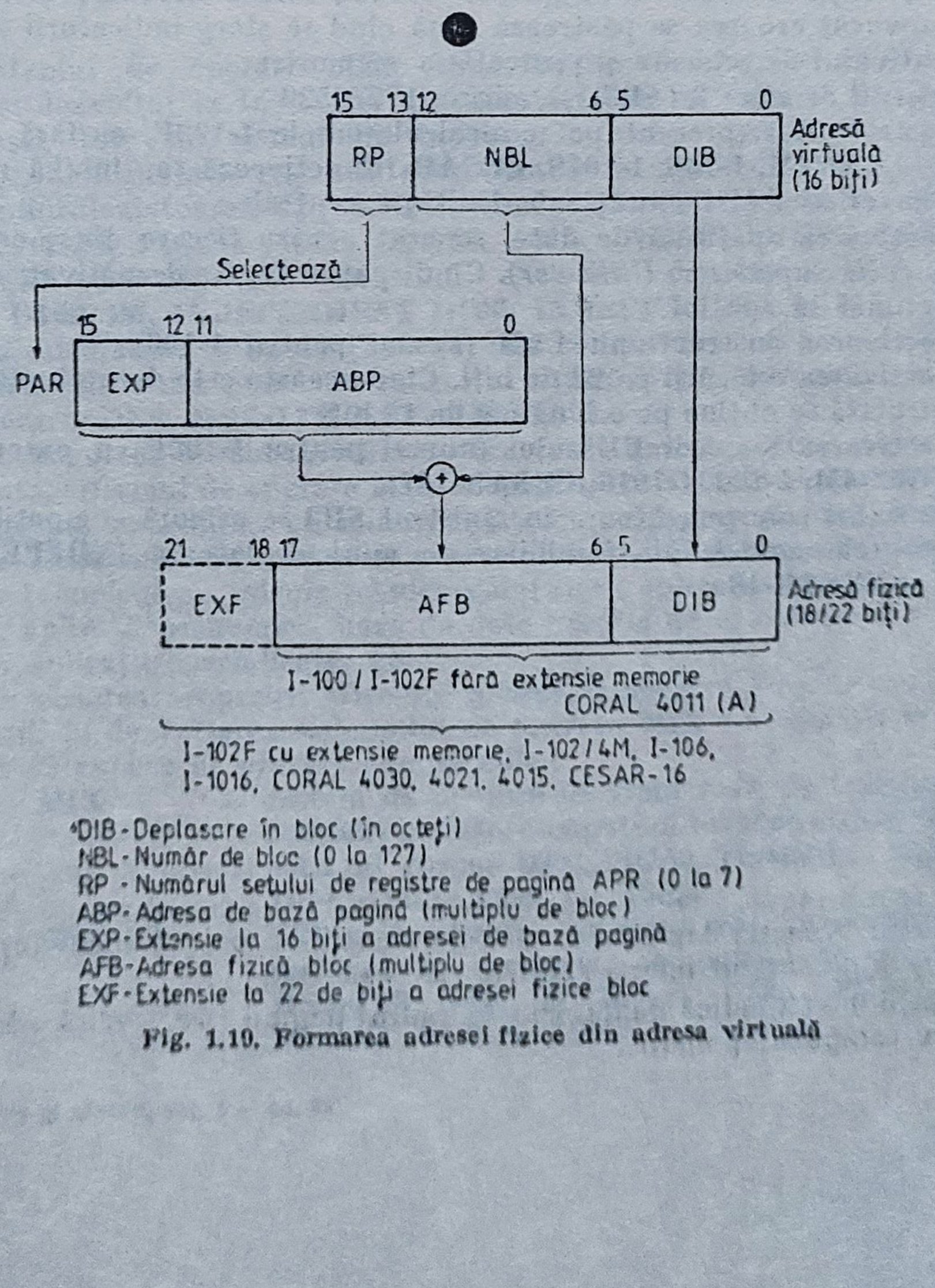


Fig. 1.10. Formarea adresei fizice din adresa virtuală

Indicatorii de eroare sînt setaţi la :

— încercarea de a scrie într-o pagină protejată la scriere ;

— încercarea de acces la o locaţie dintr-o pagină cu un număr de bloc în afara zonei autorizate de registrul **PDR** al acestei pagini ;

— încercarea de acces la o pagină nerezidentă sau neutilizată, sau de validarea relocării cu un mod ilegal în **PS.**

Cînd unul din aceşti biţi este setat, este îngheţat conţinutul biţilor 1—6 din **SRO** precum şi **SR2.**

Registrul de stare 1 (SR1) — adresa 17777574

Registrul **SR1** înregistrează cantităţile cu care se modifică registrele generale în cursul execuţiei unei instrucţiuni, ca urmare a operaţiilor de aducere a operanzilor din memorie.

La fiecare operaţie de autoincrementare sau autodecrementare, numărul registrului şi cantitatea cu care a fost modificat (în complement faţă de 2) sînt înscrise în **SR1** ; întrucît instrucţiunile pot avea 2 operanzi, sînt prevăzute două zone de înregistrare în **SR1.**

Registrul de stare 2 (SR2) — adresa 17777576

Registrul **SR2** este încărcat de Unitatea Centrală cu adresa virtuală a fiecărei instrucţiuni, programul putînd să-l acceseze numai în citire. În cazul unei erori, conţinutul său este îngheţat. Astfel, adresa virtuală a instrucţiunii care a provocat eroarea se păstrează pînă cînd se şterg indicatorii de eroare ai dispozitivului de relocare şi protecţie a memoriei.

Registrul de stare 3 (SR3) — adresa 17772520

Registrul **SR3** (prezent pe minicalculatoarele **I-102F**, cu/fără extensie memorie, **I-102/4M, I-106, I-1016, CESAR-16**) activează sau inhibă utilizarea unor caracteristici şi facilităţi ale Unităţii Centrale :

— activarea spaţiului de date, separat pentru fiecare din modurile de lucru *(Kernel, Supervizor, Utilizator)*. Cînd spaţiul **D** este dezactivat, relocarea se face numai în spaţiul **I** ;

— activarea instrucţiunii **CSM** (numai pentru **I-106**) ;

— activarea relocării pe 22 de biţi. Cînd aceasta este dezactivată, adresa fizică calculată se obţine pe o lungime de 18 biţi ;

— activarea mapării **BUS**-ului (numai pentru **I-102F** cu extensie memorie, **I-102/4M, I-106, I-1016, CESAR-16**).

Prin setări corespunzătoare în registrul **SR3** se asigură compatibilitatea între diferitele modele ale familiilor de minicalculatoare **INDEPENDENT, CORAL** şi **CESAR-16.**

2

Familia de sisteme de operare MIX/MIX-PLUS

2.1. Prezentare generală

Sistemul de operare **MIX** (*Minicomputer Interactive eXecutive*), proiectat și dezvoltat la Institutul de Cercetări pentru Tehnică de Calcul (ITC) București, este un sistem de operare în timp real, orientat pe evenimente, care permite execuția programelor în regim de multiprograme, pe toate minicalculatoarele de producție românească (**CORAL 4001, CORAL 4011, CORAL 4030, CORAL 4021, CORAL 4015, INDEPENDENT I-100, INDEPENDENT I-102F** (cu sau fără extensie de memorie), **INDEPENDENT I-102/4M, INDEPENDENT I-106, INDEPENDENT I-1016, CESAR-16,** pe minicalculatoare **PDP-11 (24—84)** sau compatibile cu acesta (**SM4/SM5**), fiind echivalent din punct de vedere al utilizării și programării cu sistemele de operare **DEC RSX-11M/RSX-11M-PLUS.**

Conceput ca un sistem unic pentru toată gama configurațiilor de calcul existente, sistemul de operare **MIX** acoperă toate solicitările generale sau specializate ale aplicațiilor cu minicalculatoare : culegere și prelucrare primară de date ; control industrial și de proces, aplicații de laborator ; comanda proceselor tehnologice ; calcule tehnico-științifice ; aplicații interactive cu multiacces ; aplicații economice, baze de date ; rețele de calculatoare și teletransmisie ; aplicații specializate, etc.

În vederea acoperirii eficiente a acestui spectru larg de configurații și aplicații, în dezvoltarea sistemului de operare **MIX** s-a urmărit crearea unei *familii de sisteme de operare* compatibile :

— **MIX** = Sistem de operare de timp real, cu multiprogramare, rezident pe disc, compatibil total cu sistemul **RSX-11M V4.1** al firmei **DEC**, SUA. Versiuni : **MIX V1.0** — 1984 ; **MIX V2.0** — 1985.

— **MIX-RT** = Sistem de operare, de timp real, cu multiprogramare, rezident în memorie, compatibil total cu sistemul **RSX-11S V4.1** al firmei **DEC**, SUA. Versiuni : **MIX-RT V1.0** — 1985.

— **MIX-PLUS** = Sistem de operare de tip real, cu multiprogramare, rezident disc, pentru minicalculatoarele de mare performanță **I-106, I-102/4M, I-1016, CORAL 4021**, compatibil total cu sistemul **RSX-11M PLUS V3.1**, al firmei **DEC**, SUA. Versiuni : **MIX-PLUS V1.0** — 1986 ; **MIX-PLUS V2.0** — 1989.

Componentele celor trei variante de sistem sînt realizate pe baza unei surse unice, înglobînd (unde este cazul) secvențe specifice, dublu condiționate.

Componentele privilegiate (Monitor, Drivere de intrare/ieșire, Interfețe operator, Task-uri sistem, Subsisteme privilegiate, Interfețe de rețea) dublu compatibile, se adaptează în mod specific (prin asamblare condiționată) celor două mari variante de sistem (**MIX** și **MIX-PLUS**).

Componentele neprivilegiate (Utilitare, Subsisteme conversaționale, Compilatoare, Module de acces **FCS/RMS**, Biblioteci, Utilitare de rețea) sînt identice pe cele două mari variante de sistem (**MIX** și **MIX-PLUS**).

Varianta de sistem rezidentă în memorie, **MIX-RT**, are componentele privilegiate identice cu cele de la sistemul de operare **MIX** (Monitor, Drivere de intrare/ieșire, Interfață operator) și componente specifice (Task-urile sistem și Modulele de acces **FCS**).

Sistemul de operare **MIX-PLUS** conține în plus o serie de componente privilegiate, inexistente în **MIX** : Subsistem de înregistrare pe discuri gemene, Subsistem de reconfigurare dinamică, Subsistem batch, etc.

Această modularitate a familiei de sisteme **MIX** permite includerea specifică a anumitor facilități (respectiv componente) **MIX/MIX-RT/MIX-PLUS**, la generarea sau reconfigurarea sistemelor, în concordanță cu cerințele particulare ale aplicației sau configurației de dispozitive periferice.

Deși oferă un număr mare de servicii, sistemele de operare **MIX** utilizează foarte puțin din resursele hardware. Ele pot fi generate (reconfigurate) să lucreze pe sisteme de calcul cu 64—128 Ko (**MIX-RT**), 192—256 Ko (**MIX**) 256 Ko — 4 Mo (**MIX, MIX-PLUS**).

În general, sistemele de operare **MIX/MIX-PLUS** se livrează utilizatorilor cu facilități și caracteristici sistem maxime (fără generare).

În funcție de aplicație, la instalarea sistemelor, prin reconfigurare, se poate selecta un număr variabil de dispozitive de I/E și componente software, sistemele fiind capabile să lucreze pe configurații mici, mijlocii, mari și foarte mari.

Sistemul de operare **MIX-RT** se poate adapta specific, prin generare, pentru aplicațiile specializate ale utilizatorilor.

2.2. Caracteristicile familiei de sisteme de operare MIX

Principalele caracteristici ale familiei de sisteme de operare **MIX** sînt următoarele :

MIX-RT

— sistem de operare complet rezident în memorie ;

MIX/MIX-PLUS

— sistem de operare rezident disc ;

MIX-RT/MIX/MIX-PLUS

— multiprogramare eficientă în timp real ;

— multitasking, cu posibilitatea creerii unor structuri complexe de relații de paternitate între task-uri ;

— planificare de task-uri bazată pe evenimente și priorități ;

— planificare de task-uri bazată pe cuante de timp ;

— sincronizare și comunicare între task-uri ;

— alocare statică și dinamică a memoriei, în partiții și subpartiții ;

— moduri de lucru Sistem (Kernel) și Utilizator (User) ;

— protecția automată a memoriei ;

— extinderea posibilităților de adresare a memoriei peste 64 Ko (pînă la 4 Mo) ;

— utilizarea bibliotecilor și blocurilor de comun, rezidente în memorie și cu acces multiplu ;

— suport pentru întreruperi sincrone (**SST**) și asincrone (**AST**) ;

— suport pentru conectarea programelor de aplicație la întreruperi externe ;

— posibilitatea recuperării complete la apariția avariilor de tensiune ;

— interfață operator — sistem (în aplicații specializate, sub **MIX-RT** aceasta poate lipsi) ;

— sistem de intrare/ieșire flexibil și modular ;

— programare independentă de dispozitivele de intrare-ieșire ;

— asigurarea protecției între programe și utilizatori ;

— reconfigurare flexibilă și dinamică a sistemelor ;

— suport pentru rețele de calculatoare.

MIX-RT

— sistem de gestiune a fișierelor pe suporturi secvențiale.

MIX/MIX-PLUS

— sisteme performante de gestiune a fișierelor pe suporturi magnetice (discuri și benzi), cu interfețe de acces tip **FCS** și **RMS** ;

— dezvoltare interactivă de programe în regim de multiacces, local sau la distanță ;

— contabilizarea resurselor sistem și utilizator ;

— limbaje multiple de comandă (**MCL, DCL**) ;

— suport pentru crearea unor limbaje interpretoare de comenzi (**CLI**) utilizator.

MIX-PLUS

— acces utilizator la spații separate de instrucțiuni (**I**) și date (**D**) ;

— acces utilizator la modul de lucru Supervizor ;

— task-uri multi-utilizator :

— *„cache“* de date pentru discuri rapide și benzi cu transfer continuu ;

— optimizarea acceselor la dispozitivele de intrare/ieșire ;

— suport pentru cataloage de fișiere (disc) cu nume ;

— acces dual la memorie și discuri magnetice ;

— suport pentru nume logice ;

— suport pentru înregistrare pe discuri gemene ;

— suport pentru encriptare parole de acces utilizator ;

— dezvoltare de programe în regim batch ;
— reconfigurare dinamică „*on-line*“ ;
— suport pentru sisteme multiprocesor (cu 1 ÷ 4 UC) ;
— facilități extinse de vectorizare a Monitorului, driverelor de intrare/ieșire și a componentelor privilegiate ;
— îmbunătățiri de performanțe.

2.3. Componentele familiei de sisteme de operare MIX

În funcție de complexitate și varianta de sistem livrată la utilizator, componentele familiei de sisteme **MIX/MIX-RT/MIX-PLUS** se împart în următoarele categorii :

— Componente de bază, parte integrantă a sistemului de operare standard ;
— Limbaje și procesoare de limbaj ;
— Subsisteme de gestiunea datelor ;
— Prelucrări distribuite în rețele de calculatoare ;
— Opțiuni de comunicație.

Majoritatea componentelor din ultimele patru categorii sînt opționale, depinzînd de aplicația utilizator și configurația specifică a acestuia.

2.3.1. Componentele de bază ale sistemelor de operare MIX

MONITOR — gestionează alocarea resurselor hardware și software și controlează întreaga activitate a sistemului ;

DRIVERE DE INTRARE/IEȘIRE — permit controlul operațiilor de transfer cu diverse dispozitive de intrare/ieșire ;

INTERFAȚA OPERATOR (**MCL**) — implementează interfața utilizator sistem, pentru controlul interactiv al dezvoltării și execuției de task-uri ;

LIMBAJUL DE COMANDĂ STANDARD (**DCL**) — implementează o interfață utilizator-sistem, de nivel înalt, pentru prelucrări interactive sau batch ;

PROCESORUL FIȘIERELOR DE COMENZI (**IND**) — permite crearea și execuția conversațională a fișierelor de comenzi operator (**MCL, DCL, CLI**) ;

PROCESORUL BATCH (**BPR**) — permite, numai sub **MIX-PLUS**, prelucrarea în loturi a unor fișiere de comenzi operator (**MCL, DCL, CLI**) și date ;

TASK-URI SISTEM — extind interfața operator-sistem, asigurînd o parte din caracteristicile standard ale sistemului de operare ;

SISTEMUL DE GESTIUNE A FIȘIERELOR — permite crearea, exploatarea și protejarea fișierelor, prin intermediul a două seturi de rutine de acces la fișiere (**FCS** și **RMS**) și a două procesoare de control pentru dispozitive : **F11ACP** (discuri magnetice), **MTAACP** (benzi magnetice) :

FCS — rutine de acces standard, optimizate pentru multiacces și prelucrări în timp real. Ele asigură acces la dispozitivele cu

structură de fișiere (discuri magnetice — format **FILES11** și benzi magnetice — format **ANSI**) sau nedefinite (fără structură de fișiere), în acces secvențial sau direct, la nivel de bloc ;

RMS — rutine de acces pentru aplicații economice și gestiune de date. Ele permit accesul secvențial sau direct la fișiere disc cu organizare secvențială, relativă sau indexat-secvențială.

SUBSISTEMUL DE ÎNREGISTRARE MESAJE OPERATOR (**COT**) — permite înregistrarea pe disc a tuturor mesajelor adresate operatorului sistem ;

SUBSISTEMUL DE ÎNREGISTRARE A ERORILOR (**ERL, ELI, RPT**) — permite înregistrarea și raportarea erorilor hardware apărute în exploatarea normală a sistemului de calcul (hardware și software) ;

SUBSISTEMUL INTEGRAT DE TESTARE (**IOX**) — permite testarea configurației de dispozitive periferice, cu ajutorul unor operații complexe de I/E ;

SUBSISTEMELE DE DEPANARE — permit depanarea programelor sistem sau utilizator :

MDS — depanator sistem pentru Monitor și drivere ;
ODT — depanator interactiv pentru programele utilizator ;
TRACE — trasor al execuției programelor utilizator ;
DEBUG — depanator simbolic, intern sau extern, pentru programele utilizator (numai sub **MIX-PLUS**) ;
PMD — program de vizualizare a memoriei atașate unui task terminat anormal ;
CDA — program de analiză a imaginilor căderilor sistem ;
NDA — program de analiză a „imaginilor“ sistem în regim distribuit (în rețea).

SUBSISTEME DE IMPRIMARE ASINCRONĂ (SPOOLING) — permit imprimarea asincronă a unor fișiere pe dispozitivele de ieșire ale configurației :

PRT — serializează accesul multiplu la imprimanta unică a unei configurații de calcul ;
QMG — permite crearea și gestionarea pe disc a unor cozi multiple de fișiere de intrare și ieșire, pentru configurațiile de calcul cu mai multe dispozitive de ieșire.

SUBSISTEMUL DE CONTABILIZARE (**ACC**) — permite înregistrarea și raportarea de informații privind utilizarea resurselor sistem și utilizator, sesiunile interactive de la terminale, măsurarea globală a performanțelor sistem ;

SUBSISTEMUL DE ÎNREGISTRARE PE DISCURI GEMENE (**SHADOW**) — permite, numai sub **MIX-PLUS**, duplicarea informațiilor (pe două discuri gemene) și asigurarea unei prelucrări continue în cazul căderii unuia dintre discuri ;

SUBSISTEME DE VIZUALIZARE DINAMICĂ A STĂRII SISTEM permit vizualizarea dinamică a unor informații de stare a sistemului :

RMD — informații privind alocarea resurselor de memorie, starea unuia sau mai multor task-uri din sistem, I/E, etc. ;

NTD — informații privind alocarea resurselor în regim distribuit (în rețea).

SUBSISTEMUL DE RECONFIGURARE ON-LINE (**CON, HRC**) — permite, numai sub **MIX-PLUS**, reconfigurarea dinamică a caracteristicilor UC și a dispozitivelor de I/E ;

UTILITARE PENTRU DEZVOLTAREA DE PROGRAME :

EDI — Editor de texte conversațional, orientat linie ;
EDT — Editor de texte conversațional, orientat bloc, cu mod de lucru ecran ;
TKB — Editor de legături ;
LBR — Bibliotecar ;
SLP — Editor de texte batch ;
ZAP —Program de corectare a „imaginilor“ task ;
PAT — Program de corectare a fișierelor intermediare, tip obiect ;
CMP — Program de comparare fișiere sursă ;
CRF — Program de afișare a referințelor încrucișate, în urma macroasamblării sau linkeditării unor programe.

UTILITARE PENTRU GESTIUNEA FIȘIERELOR ȘI VOLUMELOR :

PIP — Program de întreținere și copiere fișiere/volume ;
DMP — Program de vizualizare conținut fișiere/volume ;
VFY — Program de verificare a consistenței logice a volumelor ;
BAD — Program de identificare a blocurilor defecte disc ;
DSC — Program de salvare/restaurare a volumelor disc ;
BRU — Program de salvare/restaurare incrementală/totală a unor fișiere/volume disc ;
PRE — Program de copiere volume bandă/disc.

UTILITARE PENTRU RECONFIGURAREA SISTEMULUI :

VMR — Procesor de comenzi virtuale pentru reconfigurarea parametrilor sistem ;
MIXDSC, MIXBRU — Sisteme autoîncărcabile **MIX** pentru salvarea/restaurarea volumelor disc conținînd sisteme autoîncărcabile **MIX-RT/MIX/MIX-PLUS.**

BIBLIOTECI DE MACROINSTRUCȚIUNI ȘI RUTINE SISTEM — permit dezvoltarea completă a aplicațiilor sistem și utilizator sub sistemele de operare **MIX/MIX-PLUS.**

2.3.2. Limbaje și procesoare de limbaj

MACROASAMBLOR (**MAC**) — este limbajul de asamblare al sistemului, ce permite utilizatorilor folosirea directă a instrucțiunilor mașină și a directivelor sistem. Toate componentele sistemelor de operare **MIX-RT/ /MIX/MIX-PLUS** sînt scrise în limbaj de macroasamblare, utilizînd un set puternic și eficient de macroinstrucțiuni de programare structurată ;

FORTRAN-IV (**FOR**) — limbaj de nivel înalt orientat către calcule tehnico-științifice și inginerești. Compilatorul asociat implementează un superset al standardului **FORTRAN IV** ANSI 3.9 1966 ;

FORTRAN-77 (**F77**) — limbaj de nivel înalt, orientat atît către calculele tehnico-științifice și inginerești, cît și pe prelucrarea unor cantități mari

de date (cu posibilitatea exploatării unor fișiere **RMS** indexat-secvențiale, multi-cheie). Compilatorul asociat, ce implementează standardul **FORTRAN 77** ANSI 3.9.1978, generează un cod obiect puternic optimizat ce utilizează procesorul de virgulă mobilă prezent pe minicalculatoarele **I-102F, I-102/4M, I-106, I-1016, CORAL 4021 (DP21), DP15 și CESAR-16** ;

BASIC-PLUS **(BPL)** — limbaj de nivel înalt utilizat în aplicații interactive în domeniile științific, economic și învățămînt. Interpretorul asociat, ce extinde masiv specificațiile limbajului **BASIC** Dartmouth, include opțiuni de compilare pentru obținerea unor module obiect **BASIC-PLUS** și permite utilizarea procesorului de virgulă mobilă prezent pe minicalculatoarele **I-102F, I-102/4M, I-106, I-1016, CORAL 4021 (DP21), DP15 și CESAR-16** ;

BASIC-PLUS-2 **(BP2)** — limbaj de nivel înalt, superset al limbajului **BASIC-PLUS**, utilizat în aplicații sofisticate de culegere și prelucrare de date în domeniile științific și comercial. Compilatorul asociat permite utilizarea unor construcții structurate la nivel de bloc și a unor metode de acces **RMS** pentru adresare secvențială, relativă și indexată a fișierelor ;

COBOL **(CBL)** — limbaj de nivel înalt proiectat pentru aplicații economice și utilizat pe scară largă datorită facilităților puternice de acces la fișiere **RMS** indexat-secvențiale, multi-cheie. Compilatorul asociat, ce implementează standardul **COBOL** ANSI-74, are posibilitatea generării unui cod obiect ce utilizează procesorul comercial prezent pe minicalculatorul **I-106** ;

COBOL81 **(C81)** — limbaj de nivel înalt, superset al limbajului **COBOL** ;

CDL **(CDL)** — limbaj de nivel înalt utilizat la scrierea unor compilatoare sau interpretoare de limbaj. Compilatorul asociat este bazat pe standardul **CDL-1** ;

CPL1 **(CPL)** — limbaj de nivel înalt utilizat la scrierea unor aplicații portabile în domeniul științific sau economic. Compilatorul asociat, subset al specificațiilor **PL/1**, permite elaborarea unor aplicații independente de suportul de intrare/ieșire utilizat ;

RTL2 **(RTL)** — limbaj de nivel înalt utilizat la scrierea unor aplicații în timp real. Compilatorul asociat asigură utilizatorului un acces deplin la directivele Monitor și implementează mecanisme specifice aplicațiilor critice în timp ;

PASCAL **(PAS)** — limbaj de nivel înalt utilizat pentru dezvoltarea unor aplicații tehnico-științifice, economice și educaționale. Compilatorul asociat, ce implementează standardul **PASCAL** ISO 7185, generează un cod obiect optimizat și permite utilizarea unor metode de acces **RMS** pentru exploatarea unor fișiere secvențiale, relative și indexate ;

C **(CCC)** — limbaj de nivel înalt permițînd descrierea unor structuri moderne de control și date, utilizînd operatori simpli și puternici. Compilatorul asociat, bazat pe limbajul **C** al sistemului de operare **UNIX V7**, generează un cod obiect puternic optimizat ce utilizează procesorul de virgulă mobilă prezent pe minicalculatoarele **I-102F, I-102/4M, I-106, I-1016, CORAL 4021 (DP21), DP15 și CESAR-16** ;

ADA (**ADA**) — limbaj de nivel înalt utilizat pentru dezvoltarea unor produse de programare portabile, incluzînd trăsături ale limbajelor **ALGOL 68, PASCAL, PL/1, SIMULA**, etc. Compilatorul asociat, subset al standardului **ADA DOD 1979**, implementat în limbajul **C**, generează un cod obiect optimizat ce utilizează procesorul de virgulă mobilă prezent pe minicalculatoarele **I-102F, I-102/4M, I-106, CORAL-4021 (DP21)**, și **CESAR-16** ;

LISP (**LSP**) — limbaj de programare utilizat în cercetări de inteligență artificială și în elaborarea unor sisteme expert. Interpretorul asociat, bazat pe standardul **DM-LISP**, asigură un mediu interactiv complet pentru dezvoltarea de software în domeniul inteligenței artificiale.

În sistemele de operare **MIX/MIX-PLUS**, dezvoltarea de programe se poate face în limbajele enumerate mai sus, local sau la distanță. În cazul utilizării suportului distribuit de rețea asigurat de **MININET/MIX**, aplicațiile utilizator se pot asambla, compila și linkedita la distanță, putînd fi ulterior executate în oricare nod al rețelei de minicalculatoare.

Dezvoltarea de programe pentru sistemul **MIX-RT** se poate face, local sau la distanță, pe sisteme **MIX/MIX-PLUS**, în limbajele **MACRO, FORTRAN-IV, FORTRAN-77, RTL2, PASCAL, C** și **ADA**. Transportul local se face prin intermediul unor utilitare specifice (**FLX/OTL**). În cazul utilizării suportului distribuit de rețea, imaginile complet dezvoltate (**MIX-RT** cu programe de aplicație) pot fi transportate de pe noduri centrale **MIX/MIX-PLUS** pe noduri satelit **MIX-RT**.

2.3.3. Subsisteme conversaționale de gestiune și administrare a datelor

UTILITARE PENTRU FIȘIERE **RMS** :

RMSDEF — program de definire a atributelor unui fișier **RMS** sau parametrilor unei înregistrări ;

RMSDES — program de definire interactivă a unui fișier **RMS** ;

RMSIFL — program de inițializare a unui fișier cu organizare **RMS** indexat-secvențială ;

RMSDSP — program de vizualizare a atributelor unui fișier **RMS** ;

RMSCNV — program de conversie a fișierelor **RMS** cu diferite organizări ;

RMSBCK — program de salvare a fișierelor **RMS** ;

RMSRST — program de restaurare a fișierelor **RMS**.

SORT — program interactiv de sortare a unor date binare, EBCDIC sau ASCII, conținute în fișiere cu format secvențial, relativ și indexat-secvențial (după una sau mai multe chei) ;

DATATRIEVE (**DTR**) — subsistem interactiv de interogare și raportare, proiectat pentru un acces eficient la structuri de date conținute în fișiere **RMS** cu organizare secvențială, relativă și indexat-secvențială. Subsistemul permite regăsirea selectivă a unor informații specificate conversațional, sortarea, formatarea și actualizarea acestora, precum și generarea unor rapoarte asociate ;

FMS — subsistem conversaţional pentru interfaţarea programelor de aplicaţie cu facilităţi interactive, orientate ecran. Utilizat în conjuncţie cu programe de aplicaţii scrise în limbajele **MACRO, BASIC-PLUS-2, COBOL, COBOL-81, FORTRAN-IV, FORTRAN-77, PASCAL,** subsistemul asigură înlocuirea interfeţei tradiţionale de introducere a datelor către aplicaţie, cu video-formate compatibile **VT52/VT100.**

2.3.4. Prelucrări distribuite şi produse de comunicaţie

În scopul creşterii eficienţei de utilizare a configuraţiilor de calcul şi aplicaţiilor utilizator, familia de sisteme de operare **MIX** integrează un set complet de facilităţi de reţea şi opţiuni de comunicaţie cunoscut sub numele de **MININET/MIX** (**MINI**computers **NET**work arhitecture), compatibil cu produsul similar **DECNET/RSX-11** al firmei **DEC**, SUA.

Produsele program de comunicaţie, existente sub **MIX**, pot fi grupate în următoarele categorii :

— **MININET/MIX** — un sistem distribuit de reţea, ce permite realizarea unor reţele de minicalculatoare lucrînd sub familia de sisteme de operare **MIX-RT/MIX/MIX-PLUS.** Reţelele distribuite utilizează legături punct-la-punct şi multipunct în diverse topologii simple (stelare, ierarhice, arborescente, inelare) sau combinate ;

— **INTERNETS** — produse program utilizate pentru realizarea unor interconectări eterogene, între minicalculatoare şi calculatoare medii-mari ;

— **PACKETNETS** — produse program utilizate pentru conectarea reţelelor **MININET/MIX** la alte reţele, prin intermediul unei reţele de transport publice, cu comutare de pachete.

2.3.4.1. Sistemul distribuit de reţea MININET/MIX

Din punct de vedere arhitectural, **MININET/MIX** este organizată pe şapte nivele, fiecare asigurînd un set distinct de funcţii de reţea şi protocoale de implementare a acestor funcţii :

— Nivelul utilizator ;
— Nivelul de administrare şi gestionare a funcţiilor de reţea **(NICE)** ;
— Nivelul aplicaţiilor de reţea **(DAP)** ;
— Nivelul serviciilor de reţea **(NSP)** ;
— Nivelul de transport **(TRA)**, cu facilităţi complete de rutare adaptivă;
— Nivelul de protocolare a schimburilor **(DLC)**, utilizînd :
  — protocoale mod „byte“ — **MDLC, BSC** ;
  — protocoale mod „bit“ — **HDLC** (X.25 nivel 2).
— Nivelul legăturii fizice **(DDM)**.

Facilităţile importante asigurate de reţeaua **MININET/MIX** sînt :

— Comunicarea între task-uri ;
— Acces la resurse în reţea :
  — partajarea dispozitivelor de intrare/ieşire ;
  — transferul fişierelor la distanţă ;
  — accesul fişierelor la distanţă ;
  — terminale virtuale.

— Controlul task-urilor la distanță ;
— Încărcarea/evacuarea task-urilor la/de la distanță ;
— Încărcarea sistemelor la distanță ;
— Comunicarea între terminale ;
— Accesul direct la liniile de comunicație ;
— Gestiune și administrare de rețea.

Conceput ca un produs distribuit de rețea, **MININET/MIX** asigură legăturile de transmisie pe linii de comunicație asincrone și sincrone, în rețele omogene de minicalculatoare conduse de sisteme de operare **MIX-RT/MIX/MIX-PLUS** sau **DECNET-11M/S V3.1/V4.2, DECNET-11M PLUS V1.1/V2.2** și **DECNET/VAX.**

Accesul la rețeaua **MININET/MIX** este permis programelor utilizator scrise în limbajele **MACRO, FORTRAN-IV, FORTRAN-77, PASCAL, COBOL, COBOL-81, BASIC-PLUS-2.**

Pentru realizarea arhitecturii de rețea **MININET/MIX**, au fost concepute *variante* specifice familiei de sisteme **MIX :**

— **MININET/MIX** = sistem distribuit de rețea sub **MIX V.2.0**, compatibil cu pachetul de rețea **DECNET-11M V3.1/V4.2** al firmei **DEC**, SUA ;
— **MININET/MIX-PLUS** = sistem distribuit de rețea sub **MIX-PLUS V1.0/V2.0**, compatibil cu pachetul de rețea **DECNET-11M PLUS V2.0/V2.2** al firmei **DEC**, SUA ;
— **MININET/MIX-RT** = sistem distribuit de rețea sub **MIX-RT V2.0**, compatibil cu pachetul de rețea **DECNET-11S V3.1** al firmei **DEC**, SUA ;
— **MININET/DLX** = sistem minimal de asigurare a comunicațiilor intercalculatoare, compatibil cu pachetul de comunicații **DLX-11** al firmei **DEC**, SUA.

2.3.4.2. Interconectări eterogene sub MIX (INTERNETS)

— **MIX 3271 PE** = asigură comunicarea interactivă între programe, prin emularea terminalului interactiv **IBM 3270** (**BSC** interactiv) ;
— **MIX 2780/3780 PE** = asigură transfer de fișiere la distanță prin emularea terminalului greu **IBM 2780/3780** (**BSC** batch).

2.3.4.3. Interfețe la rețelele publice cu comutare de pachete

În scopul realizării unei interconectări între rețeaua de minicalculatoare **MININET/MIX** și alte rețele eterogene (de minicalculatoare sau calculatoare medii-mari), arhitectura **MININET/MIX** asigură interfețe de acces la rețele publice cu comutare de pachete, bazate pe standardul **X.25.**

Interfețele de acces la rețea cunoscute sub numele de **NETX25**, asigură o conectare punct la punct, fizică și logică între două noduri terminale (**DTE**), prin intermediul unei rețele cu comutare de pachete, formată din mai multe noduri intermediare (cu facilități de rutare, **DCE**).

Bazată pe arhitectura **MININET/MIX**, **NETX25** asigură toate facilitățile de prelucrări distribuite în rețele deschise, cu comutare de pachete.

2.4. Evoluția și starea curentă a familiei de sisteme de operare MIX/MIX-PLUS

Apărut la începutul anului 1984, prin comasarea caracteristicilor fostelor sisteme de operare **MININET/MINOS** și **AMS**, sistemul de operare **MIX V2.0** este compatibil cu **RSX-11M V4.1** (cu o serie de caracteristici suplimentare specifice **RSX-11M PLUS V2.1**), avînd la bază o sursă unică **MIX/MIX-PLUS.** În luna iulie 1986, sistemul de operare **MIX V2.0** a fost *„înghețat"*, eliminîndu-se ulterior numai anomaliile de funcționare, toate dezvoltările ulterioare (includere noi procesoare, noi echipamente periferice, noi componente, noi facilități software, etc.) făcîndu-se numai pe sistemul de operare **MIX-PLUS.**

Trebuie menționat faptul că trecerea aplicațiilor utilizator neprivilegiate de pe **MIX** pe **MIX-PLUS** se face fără modificări, iar migrarea aplicațiilor sistem și a driverelor utilizator se face relativ ușor, documentația comună de utilizare scoțînd în evidență particularitățile și avantajele **MIX-PLUS** față de sistemul **MIX.**

Conceput inițial ca un sistem dedicat calculatoarelor **I-106**, în trimestrul 3 al anului 1986 s-a reușit adaptarea **MIX-PLUS V1.0** și pe minicalculatoarele fără hardware specializat (**I-102F** cu extensie de memorie, **CORAL 4021, DP21, DP15**). În urma acestei adaptări nu se mai justifică livrarea în continuare, pe aceste calculatoare, a sistemului de operare **MIX V2.0.**

Introducerea în fabricație a noii versiuni de minicalculator **I-102/4M** (trimestrul I/1987), ce include spații de date și instrucțiuni separate, a făcut posibilă adaptarea (în anul 1987), pe acest calculator a unui sistem de operare **MIX-PLUS** similar celui de la **I-106**, ceea ce duce la mărirea substanțială a performanțelor de ansamblu ale acestui calculator pentru memorii mari (2—4 Mo).

În prezent există și se pot livra beneficiarilor interni și externi două variante mari de sistem :

— **MIX-PLUS** *fără spații* **I** *și* **D** — pentru minicalculatoarele **CORAL 4030, 4021, DP21, DP15, I-102F** cu extensie de memorie ;

— **MIX-PLUS** *cu spații* **I** *și* **D** — pentru minicalculatoarele **I-106** și **I-102/4M.**

Ambele variante de sistem includ adaptarea automată la caracteristicile unităților centrale și dispozitivelor periferice.

În perioada 1988—1989, s-au desfășurat importante lucrări de elaborare a ultimei versiuni de sistem **MIX-PLUS V2.0** (omologată internațional, sub numele de **MFOS V2.0**, în decembrie1989, la București), compatibilă cu sistemul de referință **RSX-11M PLUS V3.1.** O bună parte din caracteristicile

acestui sistem (prezentate în diversele capitole ale vol. 1), constituie concepte noi, inovatoare, ce micşorează diferenţele de utilizare între sistemele de operare pe 16 biţi şi cele pe 32 de biţi (compatibile **VAX/VMS**).

2.5. Caracteristici specifice ale sistemului de operare MIX-PLUS V2.0

Caracteristicile noi ale ultimei versiuni de sistem **MIX-PLUS V2.0** sînt prezentate pe scurt în continuare :

NOI FACILITĂŢI MONITOR

Pornind de la o serie de constatări şi necesităţi puse în evidenţă de sistemele de operare **MIX V2.0** şi **MIX-PLUS V1.0**. Monitorul sistemului de operare **MIX-PLUS V2.0** este structurat în 5 părţi :

— Nucleul sistemului de operare, rezident disc ;
— Zona primară cu alocare dinamică de memorie („*primary pool*"), mapată în spaţiul de date (**D**) sistem avînd o dimensiune de 56 Kocteţi ;
— Zona secundară cu alocare dinamică de memorie („*secondary pool*"), alocată în afara Nucleului, de dimensiune mare (practic limitată de memoria fizică disponibilă) şi extensibilă dinamic ;
— Zonele de comun ale Monitorului (5) — ce conţin directivele sistem (**DR1MIX**, . . ., **DR4MIX**) şi rutinele de suport „caching" ;
— Zona de vectorizare a punctelor de intrare şi locaţiilor globale ale Nucleului (**MIXVEC**) ;
— Extensii încărcabile ale Monitorului, excluse din partea rezidentă :

— **MDS**	—	depanator sistem interactiv ;
— **SEC**	—	extensor al pool-ului secundar ;
— **CDA***xy*	—	module de vidare a memoriei în caz de cădere sistem ;
— **PAR**	—	tratare derute de paritate la memorie ;
— **POWER**	—	tratare cădere de tensiune ;
— **SHADOW**	—	suport pentru înregistrare pe discuri gemene.

Monitorul include de asemenea o serie de extensii importante accesibile programelor utilizator :

— suport pentru accesul vectorizat la rutinele şi locaţiile globale ale Nucleului — realizîndu-se independenţa task-urilor sistem şi utilizator, privilegiate, de varianta particulară de Nucleu existentă în sistem ;

— suport pentru nume logice, incluzînd facilităţi de asignare (logic-logic, logic-fizic), translatare, conversie, directive Monitor şi extensii masive **FCS** şi **RMS** ;

— noi directive Monitor : asignare canal, creare/ştergere/translatare/translatare recursivă nume logice, citire director implicit, testare caracteristici sistem şi task, parsare nume fişier extins **FCS/RMS** ;

— suport pentru „buferarea" datelor de I/E disc („*data caching*"), reducînd numărul de cereri de I/E fizice la discurile magnetice ;

— suport pentru directoare disc cu nume, etc.

NOI FACILITĂȚI ALE DRIVERELOR DE INTRARE/IEȘIRE

Driverele de I/E concepute ca extensii ale Monitorului sistemului de operare **MIX-PLUS V2.0** au fost proiectate astfel încît să asigure independența acestora de configurația de I/E și unificarea cu configurațiile similare ale altor minicalculatoare **INDEPENDENT** și **CORAL**:

— acces vectorizat la rutinele și locațiile globale ale Nucleului — realizîndu-se independența și unificarea acestora, indiferent de tipul de sistem (**MIX-RT V2.0** sau **MIX-PLUS V2.0**;

— suport generalizat pentru I/E „buferate“ (asincrone);

— generalizarea și maximizarea driver-ului de terminal;

— suport pentru rețele locale (**LAT**) la nivelul driver-ului de terminal;

— includerea de noi dispozitive specifice minicalculatoarelor românești.

NOI FACILITĂȚI ALE SUBSISTEMELOR PRIVILEGIATE

În general, subsistemele privilegiate ale sistemului de operare **MIX-PLUS V2.0** au suferit modificări în două direcții:

— acces vectorizat la rutinele și locațiile globale ale sistemului (**RMD MTAACP, COT**, etc.) — asigurîndu-se indepenența acestora de Nucleu;

— extensii specifice (**VMR**, Subsistem înregistrare erori, Configurator **RMD**, etc.) în toate subsistemele privilegiate.

NOI FACILITĂȚI ALE LIMBAJELOR DE COMANDĂ MCL ȘI DCL

Interfața operator-sistem (incluzînd limbajele de comandă *MCL* și *DCL*), conține o serie de extensii importante pentru creșterea protecției între aplicațiile utilizator și sistem și pentru acoperirea noilor funcții Monitor:

— încărcarea descărcarea extensiilor vectorizate ale Monitorului;

— declararea și utilizarea unei zone de „buferare“ a datelor de I/E disc *(cache)*;

— setări și vizualizări de caracteristici specifice noilor extensii Monitor: nume logice, directoare cu nume, etc.;

— definirea și utilizarea numelor logice;

— posibilități de lucru cu seturi multiple de caractere (seturi naționale)

— parole de acces encriptate automat;

— extensii ale procesorului de comenzi indirecte;

— extensii ale funcțiilor extinse operator.

NOI FACILITĂȚI ALE SISTEMELOR DE GESTIUNEA FIȘIERELOR FCS ȘI RMS

Sistemele de gestiune a fișierelor (**FCS** și **RMS**) sub **MIX-PLUS V2.0** asigură lucrul cu fișiere de date organizate logic pe discuri (**FILES-11 ODS 1/2**) și benzi magnetice (**ANSI** nivelul 3).

Principalele extensii aduse celor două sisteme de gestiune a fișierelor sînt următoarele:

— format extins al specificatorului de fișier (incluzînd nume logice);

— parsare implicită sau explicită a numelor logice ;
— parser semantic **CSI$4** ;
— modificări în parametrii de apel macro **FCS** şi **RMS** ;
— noi formate pentru etichetele **VOL1, HDR3** ;
— număr zecimal pentru versiunea fişierelor, etc.

Sistemul de gestiune a înregistrărilor (**RMS**) precum şi formatele volumelor disc şi bandă, asigură o compatibilitate de jos în sus cu sistemul de operare **MIX/VMS** (pentru minicalculatoarele româneşti pe 32 de biţi).

NOI FACILITĂŢI ALE UTILITARELOR STANDARD

Utilitarele standard ale sistemului de operare **MIX-PLUS V2.0** au fost extinse pentru a suporta atît noile caracteristici UC şi noile dispozitive periferice ale minicalculatoarelor româneşti, cît şi noile facilităţi Monitor :

— acces la formatul extins al specificatorului de fişier, pentru toate utilitarele se folosesc **FCS** pentru citirea şi parsarea liniei de comandă ;
— suport pentru nume logice în **MACRO** ;
— extensii diverse ale Editorului de legături (**TKB**), inclusiv o nouă metodă de mapare rapidă şi directă a segmentelor rezidente în memorie (*Fast map*) ;
— extensii ale utilitarului de arhivare/restaurare suporturi/fişiere (**BRU**) ;
— extensii ale utilitarelor **RMS**, etc.

2.6. Caracteristici generale şi specifice ale sistemului de operare MIX-RT V2.0

Pornindu-se de la caracteristicile tehnice ale noului minicalculator **I-1016**, laboratorul de „*Sisteme de operare pentru minicalculatoare*“ (**F20**) din ITC, Bucureşti, şi-a propus, în anul 1989, pe baza experienţei acumulate, să realizeze un sistem de operare (**MIX-RT V2.0**) care să răspundă atît cerinţelor tehnice ale minicalculatorului, cît şi cerinţelor realizatorilor şi beneficiarilor de informatică industrială.

Pentru definirea sistemului **MIX-RT V2.0** s-au luat în considerare sistemele existente în familia de sisteme de operare **MIX** şi tendinţele lor de dezvoltare, rezultînd următoarele concluzii :

— sistemul trebuie adaptat la configuraţia hardware, propusă de realizatorul hardware, specifică utilizărilor în medii industriale ;
— sistemul trebuie să fie combatibil cu sistemele de operare existente din familia de sisteme **MIX** ;
— sistemul trebuie să asigure o compatibilitate deplină cu noile versiuni ale sistemului de operare **RSX-11M PLUS** ;
— sistemul trebuie să fie capabil să preia aplicaţii scrise pentru minicalculatoarele de fabricaţie curentă.

Pornind de la aceste considerente, principalele caracteristici ale sistemului de operare **MIX-RT V2.0** sînt :

— compatibilitate deplină cu sistemul de operare **MIX-PLUS V2.0** ;
— sistem de operare rezident disc sau rezident în memorie ;

— multiprogramare eficientă în timp real ;
— multitasking, cu posibilitatea creerii unor structuri complexe de relații de paternitate între task-uri ;
— planificare de task-uri bazate pe evenimente, priorități și cuante de timp ;
— sincronizare și comunicare între task-uri prin :
— indicatori comuni/globali de evenimente ;
— cutii poștale ;
— zone partajate de memorie ;
— fișiere partajate ;
— alocare statică și dinamică a memoriei, în partiții și subpartiții ;
— extinderea posibilităților de adresare a memoriei pînă la 4 Mocteți ;
— asigurarea protecției între programe și utilizatori ;
— acces utilizator la spații separate de instrucțiuni (I) și date (D) ;
— moduri de lucru *Kernel*, *Utilizator* și *Supervizor* ;
— utilizarea bibliotecilor și blocurilor de comun, rezidente în memorie și cu acces multiplu ;
— suport pentru întreruperi sincrone (SST) și asincrone (AST) ;
— suport pentru conectarea aplicațiilor la întreruperi externe ;
— task-uri multiutilizator ;
— facilități extinse de vectorizare a Monitorului, driverelor de intrare/ieșire și a componentelor privilegiate ;
— suport pentru nume logice ;
— sistem de intrare/ieșire flexibil și modular ;
— optimizări de acces la dispozitivele de intrare/ieșire ;
— suport pentru înregistrarea pe discuri gemene ;
— sisteme performante de gestiune a fișierelor pe suporturi magnetice (discuri și benzi), cu interfețe de acces tip **FCS** și **RMS** ;
— suport pentru cataloage de fișier (disc) cu nume ;
— contabilizarea resurselor sistem și utilizator ;
— limbaje multiple de comandă (**MCL** și **DLC**) ;
— suport pentru crearea unor limbaje interpretoare de comenzi (**CLI**) utilizator ;
— reconfigurare flexibilă și dinamică on-line ;
— dezvoltarea interactivă de programe în regim de multiacces și batch, local sau la distanță (prin preluarea componentelor specifice de la **MIX-PLUS V2.0**) ;
— suport pentru limbaje și subsisteme de administrare a datelor (prin preluarea componentelor specifice de la **MIX-PLUS V2.0**) ;
— suport pentru rețele de calculatoare ;
— compatibilitate cu modelele de firmă **RSX-11M PLUS V3.1** și **DECNET-11M PLUS V4.2.**

REZIDENȚA ÎN MEMORIE A SISTEMULUI DE OPERARE **MIX-RT V2.0**

Întrucît sistemul de operare **MIX-RT V2.0** este destinat exploatării în condiții de eficiență maximă a minicalculatorului **I-1016** (timp de răspuns mic în condiții industriale, grad mare de fiabilitate, dispozitive periferice specializate, dedicate : absența disc mare capacitate, suporturi secvențiale

de încărcare, etc.), rezidența în memorie a sistemului de operare constituie o soluție viabilă și extrem de necesară.

Versiunea de sistem rezidentă în memorie are următoarele caracteristici :

— Monitorul sistemului de operare — complet rezident în memorie, cu caracteristici identice cu cele ale versiunii rezidente disc (incluzînd însă toate extensiile în memorie) :

— Driverele de intrare/ieșire pentru configurația hardware completă, identice (ca facilități) cu cele ale versiunii rezidente disc, dar cu baze de date modificate :

— toate terminalele sînt privilegiate, simulîndu-se sesiuni deschise ;

— toate dispozitivele ce suportă structură de fișiere (discuri și benzi) au accesul validat (**/FOR**), dar fără control **ACP** (nu pot fi exploatate utilizînd sistemele de gestiune a fișierelor **FCS** și **RMS**). Programele utilizator sînt responsabile de organizarea informației pe aceste suporturi ;

— Interfața operator, cu toate facilitățile versiunii rezidente disc, va fi specifică variantei rezidente în memorie. Ea cuprinde un număr de patru task-uri cooperante (**MCD, MCR, SET, LST**), concepute ca programe cu segmente rezidente în memorie ;

— Programul de încărcare în memorie (**OTL**) a diverselor programe utilizator ce fac parte din aplicația specializată. El combină funcțiile de instalare (**INS**) cu cele de fixare program (**FIX**), cu observația că **OTL** este capabil să încarce în memorie atît programe nesegmentate cît și programe segmentate (cu segmente rezidente în memorie), regiuni de comun, programe ce se leagă de regiunile de comun, etc. **OTL** realizează încărcarea unor imagini executabile în format **RT-11** (de pe discuri magnetice) sau **DOS** (de pe banda magnetică). Plasarea programelor utilizator pe suport extern, în format **RT-11** sau **DOS**, se face utilizînd utilitarul **FLX** de pe sistemele de operare **MIX/MIX-PLUS** rezidente disc :

— Încărcătorul sistem (**LDR**) — apelat opțional (de către **OTL**), pentru încărcarea nesecvențială (de pe disc) a unor programe executabile cu structură complexă (spații **I** și **D**, etc.) ;

— Programul de salvare (**SIP**) — utilizat pentru obținerea de imagini autoîncărcabile ale sistemului de operare (incluzînd programele utilizator aflate în memorie la momentul salvării). Suporturile de salvare (benzi, discuri magnetice) sînt tratate secvențial ;

— Programe de configurare (**CON, HRC**) — utilizate pentru punerea „on-line" a dispozitivelor periferice ce urmează a fi utilizate.

Configurarea versiunii rezidente în memorie a sistemului de operare **MIX-RT V2.0** se realizează tot cu ajutorul utilitarului **VMR** (modificat și extins), ca și în cazul sistemului rezident disc. În ambele cazuri se pornește de la aceeași imagine disc Monitor (**MIXRT. TSK**), pe care se construiește sistemul (se crează partiții, se încarcă drivere și programe ce urmează a fi părți componente ale sistemului).

O particularitate deosebită, în cazul configurării versiunii rezidente în memorie, o constituie încărcarea cu fixare a programelor (sistem și utilizator),

în imaginea sistem de pe disc (comenzile **INS** și **FIX** ale **VMR**). Menționăm că tehnica segmentării programelor (utilizînd segmente rezidente în memorie) este esențială pentru utilizarea eficientă a versiunii rezidente în memorie a sistemului **MIX-RT V2.0**.

Salvarea imaginii disc configurată pe un suport cu utilizare secvențială se face prin comanda **VMR SAV**. De remarcat faptul că aplicația utilizator poate fi încărcată în imaginea sistem fie static (utilizînd utilitarul **VMR**) sub controlul unui sistem de operare rezident disc, fie dinamic (utilizînd utilitarul **OTL**, la un moment ulterior lansării în execuție a versiunii rezidente în memorie).

Realizarea versiunii rezidente în memorie a sistemului de operare **MIX-RT V2.0** constituie o abordare originală a problemei rezidenței în memorie, în condițiile coezistenței sale cu versiunile rezidente disc.

3

Concepte relative la task-uri, evenimente și planificarea task-urilor

3.1. Multiprogramare

Multiprogramarea este definită ca utilizarea concurentă a resurselor sistem, de către două sau mai multe programe (task-uri) sub controlul unei singure Unități Centrale.

Întrucît, în cazul execuției unor programe, apar diferențe între timpii solicitați pentru transferul de date în — și din — sistem și timpul necesar prelucrării acestora, se produc goluri (timpi morți) în ocuparea tuturor resurselor sistem. Multiprogramarea constituie o cale de îmbunătățire a performanțelor generale ale sistemului (hardware și software) prin constituirea unor șiruri de așteptare la resurse. Cererea este stimulată de prezența în memorie a mai multor programe așteptînd eliberarea de resurse. Multiprogramarea este realizată prin multiplexarea programelor concurente în golurile de ocupare ale sistemului de calcul.

Deoarece în configurațiile cu o singură Unitate centrală numai un singur program poate avea la un moment dat controlul acestuia, concurența aparentă este realizată împreună cu alte resurse sistem, în special dispozitivele de intrare/ieșire, ce pot lucra în paralel.

O multiprogramare eficientă este realizată numai în cazul în care mai multe programe rezidă în memorie simultan, neputîndu-și continua execuția deoarece așteaptă terminarea unor operații de intrare/ieșire, sînt sincronizate cu execuția altor task-uri sau sînt blocate pînă la apariția unor evenimente externe nepredictibile.

În acest caz, pe durata blocării unuia sau mai multor programe, sistemul de operare **MIX** permite altui program utilizarea Unității centrale (execuția de instrucțiuni).

Implementarea multiprogramării duce în general la creșterea costului general al sistemului, atît prin creșterea necesarului de memorie cît și prin dezvoltarea sistemelor de programare în sine. Aceste costuri sînt justificate, însă, pe deplin, în cazul sistemului de operare **MIX**, atît datorită cerințelor

aplicațiilor proprii MIX (aplicații de conducere de procese cu timp de reacție minim, aplicații dependente de timp nerealizabile fără exploatarea paralelismului inerent multiprogramării, aplicații cu sincronizare între programe), cît și creșterii raportului performanțe-cost sistem, obținut în sistemele cu multiprogramare.

Figurile 3.1 și 3.2 ilustrează avantajele multiprogramării față de execuția secvențială a programelor. Figura 3.1 ilustrează secvența în care se desfășoară diferitele aplicații efectuate de patru programe executate secvențial. Execuția programelor **A, B, C** și **D** într-un sistem fără multiprogramare, duce la execuția unui program după altul. Programul **A** citește un bloc de informații de pe bandă magnetică, le prelucrează și le afișează pe un display. Programul **B**

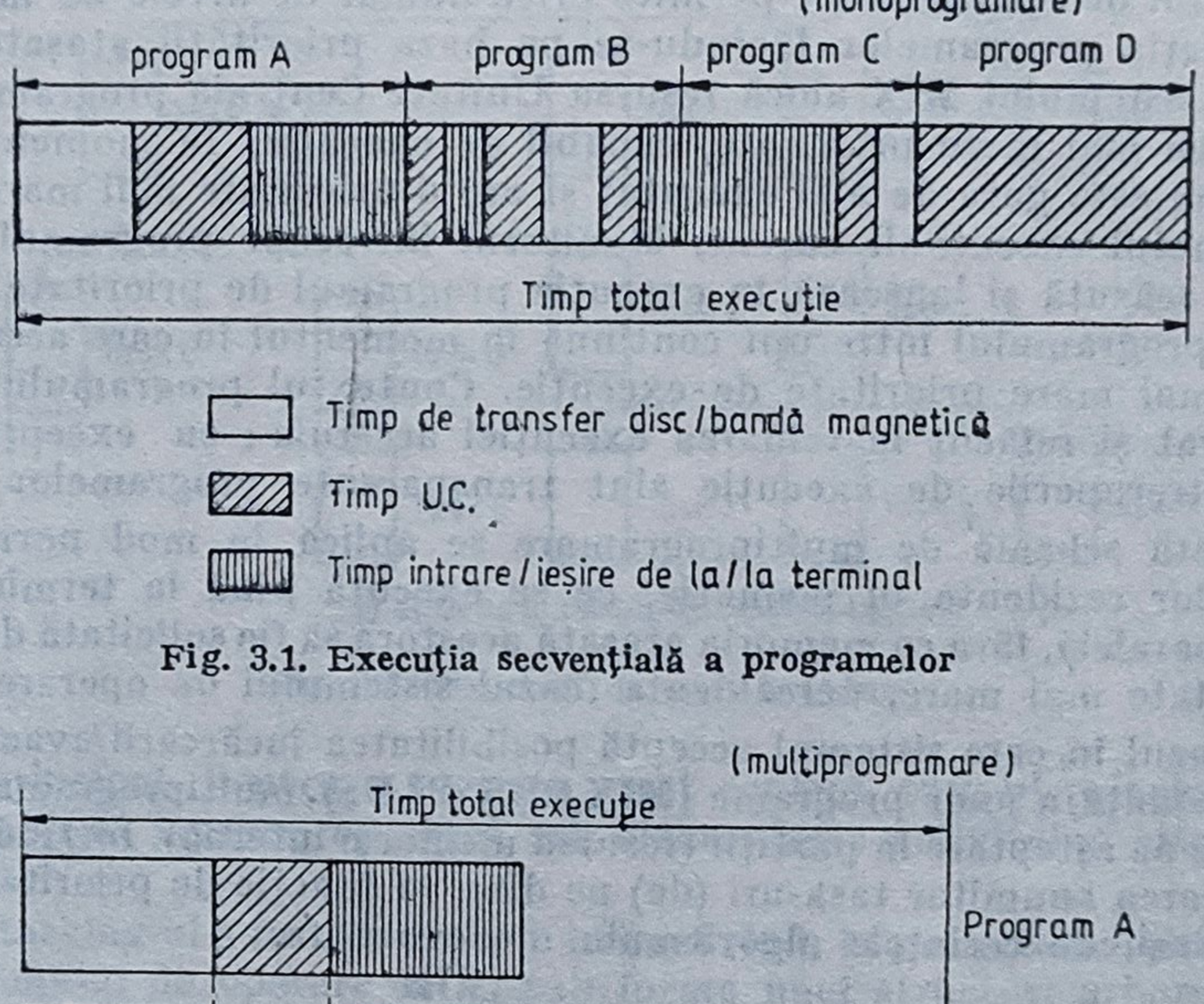


Fig. 3.1. Execuția secvențială a programelor

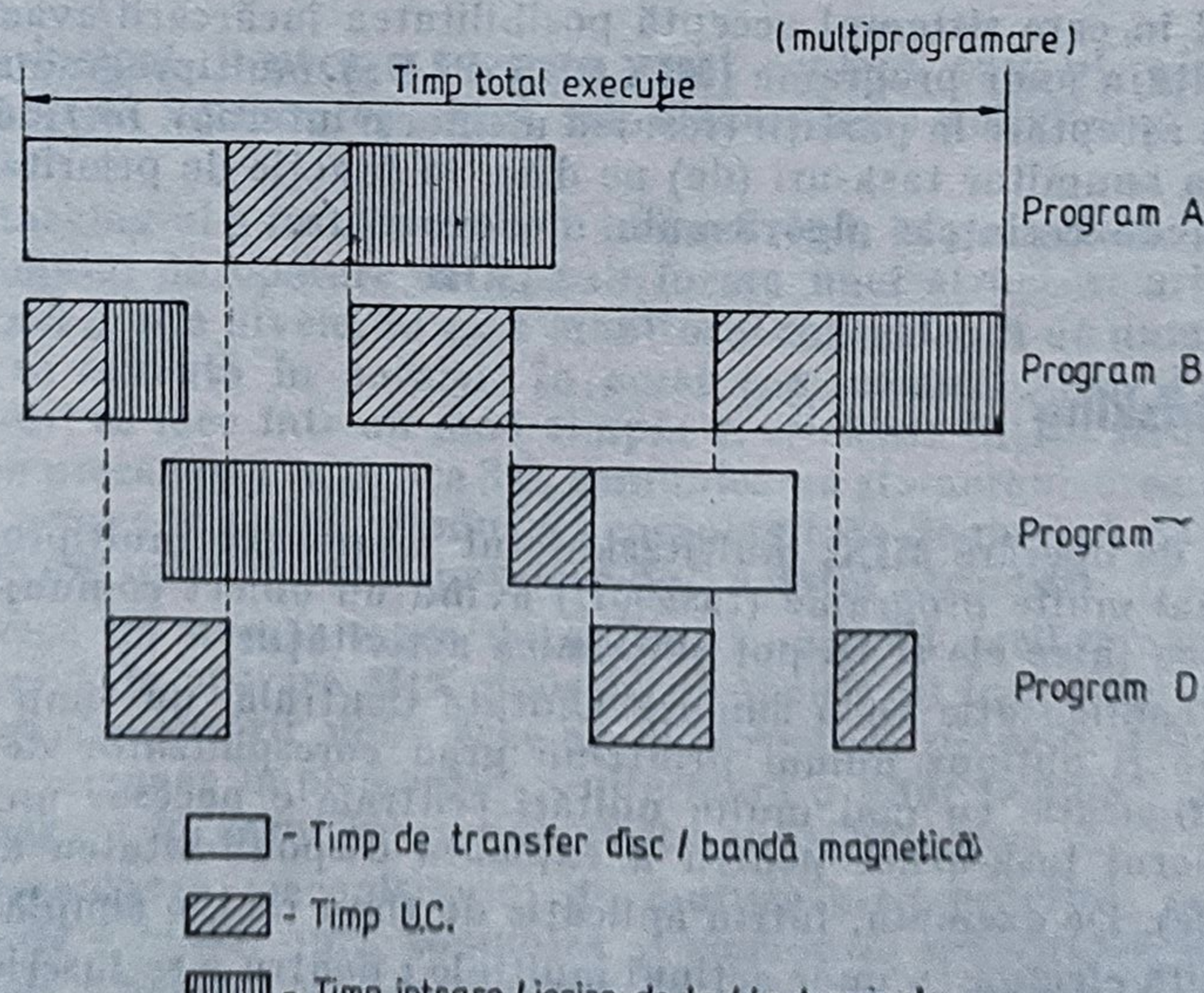


Fig. 3.2. Execuția concurentă a programelor (multiprogramare)

efectuează cîteva calcule, afişează un mesaj pe display, mai efectuează cîteva calcule, citeşte informaţii de pe un disc şi afişează rezultatul pe display. Programul **C** citeşte informaţiile introduse la display, le prelucrează şi scrie rezultatul pe bandă magnetică. Programul **D** efectuează numai calcule, fără operaţii de intrare/ieşire. În felul acesta, în timp ce unele părţi din sistem sînt ocupate, cum ar fi dispozitivele periferice, alte părţi, cum ar fi Unitatea Centrală, au timpi morţi.

Figura 3.2 ilustrează o secvenţă posibilă a operaţiilor efectuate de cele patru programe ce se execută în paralel într-un sistem cu multiprogramare de tipul **MIX**. Cele patru resurse implicate (disc, bandă magnetică, unitate centrală şi display) sînt în cea mai mare parte a timpului active ; execuţia concurentă a celor patru programe durează mai puţin decît execuţia secvenţială pe acelaşi sistem de calcul.

Sistemul de operare **MIX** permite orice număr de nivele de multiprogramare, selecţia programelor făcîndu-se pe baza priorităţii ataşate acestora. Monitorul sistemului **MIX** alocă resursa Unitate Centrală programului (task-ului) cu cea mai mare prioritate, capabil de execuţie. În momentul în care un program este gata de a fi executat şi are o prioritate mai mare decît cea a programului executabil curent, Monitorul întrerupe programul de prioritate mai scăzută şi lansează în execuţie programul de prioritate mai mare. Execuţia programului întrerupt continuă în momentul în care acesta are din nou cea mai mare prioritate de execuţie. Contextul programului întrerupt este păstrat şi refăcut la reluarea execuţiei acestuia ; cu excepţia timpilor morţi, întreruperile de execuţie sînt transparente programelor utilizator.

Această schemă de multiprogramare se aplică în mod normal numai programelor rezidente în memorie, ce se execută pînă la terminare într-o manieră paralelă, fără ca memoria ataşată acestora să fie solicitată de programe de prioritate mai mare, nerezidente (cazul sistemului de operare **MIX-RT**).

În cazul în care sistemul acceptă posibilitatea încărcării/evacuării locale sau la distanţă a unor programe (**MIX/MIX-PLUS**), multiprogramarea rezolvă şi cererile de aşteptare la partiţii (resursa memorie internă), forţînd evacuarea /reîncărcarea anumitor task-uri (de) pe disc, în funcţie de prioritatea alocată acestora sau de cerinţele algoritmului de evacuare.

3.2. Multitasking

În sistemul de operare **MIX**, multitasking-ul reprezintă multiprogramarea a două sau mai multe programe (task-uri) avînd un obiect comun, task-uri ce pot comunica între ele şi îşi pot sincroniza activităţile.

Într-o configuraţie cu o singură Unitate Centrală, un timp de răspuns optim poate fi obţinut numai printr-un grad corespunzător de paralelism (într-o configuraţie cu mai multe unităţi centrale e necesar un paralelism şi în interiorul task-urilor pentru a exploata disponibilitatea acestor Unităţi centrale). De exemplu, într-o aplicaţie de timp real, o simplă cerere operator necesită efectuarea unor acţiuni multiple ; pentru a se înscrie în limitele de răspuns, prelucrarea acţiunilor se face în paralel, sub controlul unui sin-

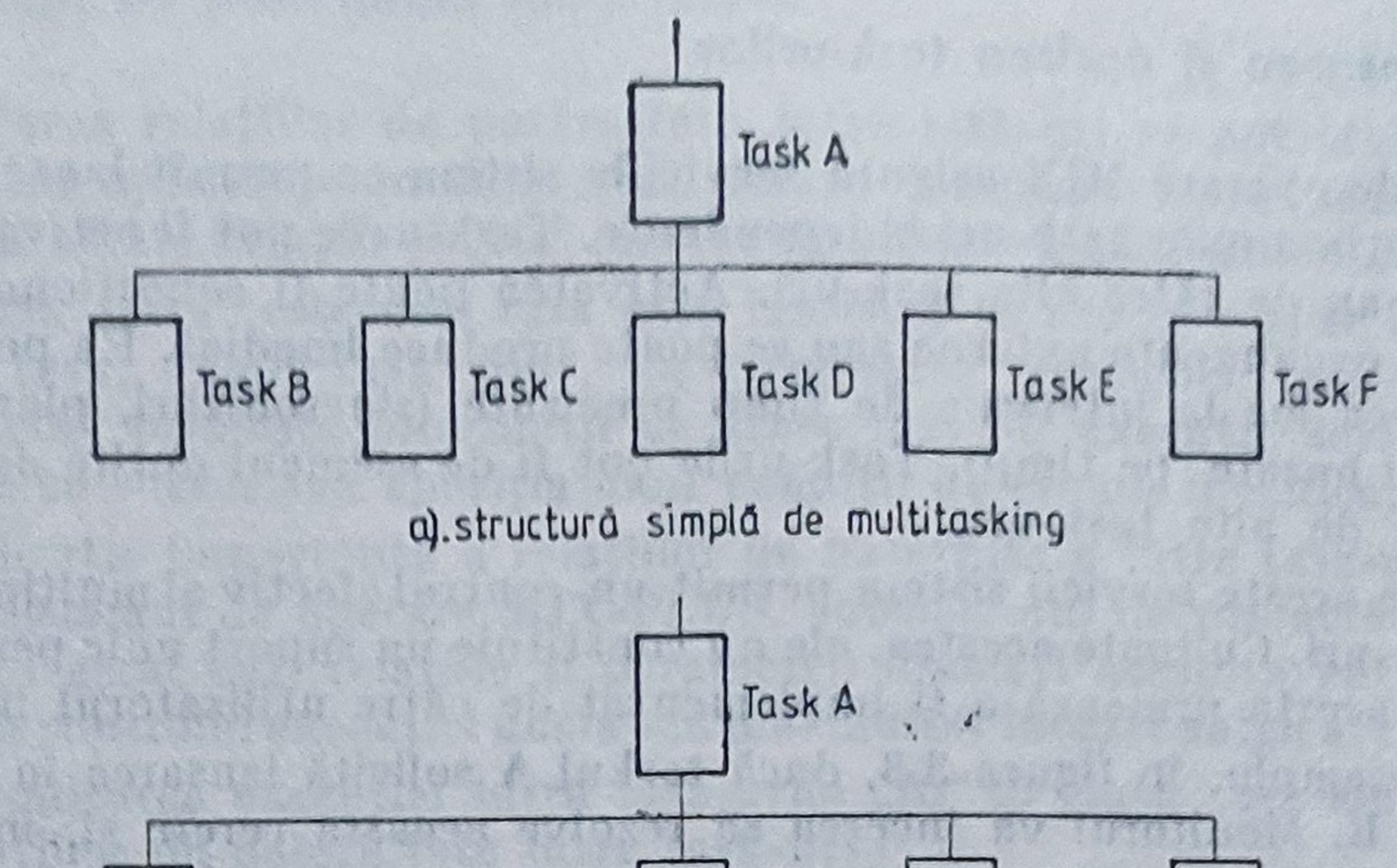


Fig. 3.3. Structuri de multitastking

gur task principal. Pentru a exercita acest control, task-ul principal trebuie să fie capabil să lanseze, oprească sau sincronizeze activitatea altor task-uri din sistem.

Multitasking-ul a fost utilizat, în primul rînd, în proiectarea și implementarea sistemului de operare **MIX**, sub forma unei structuri arborescente pe două sau mai multe nivele, în care Monitorul controlează un număr de task-uri sistem ce se execută în paralel. În acest caz, comunicarea și sincronizarea între task-uri se face într-un mod simplu și eficient. Într-o structură complicată, ca cea prezentată în figura 3.3, comunicarea și sincronizarea între task-uri devine un proces complex, gradul de complexitate depinzînd de transparența cu care apare utilizatorului paralelismul acestei structuri.

Deoarece un număr mare de aplicații necesită facilitarea de multitasking, sistemul de operare **MIX** pune la dispoziția utilizatorilor și primitivele necesare implementării unor scheme și structuri proprii. Aceste primitive permit de asemenea implementarea ulterioară a unor facilități mai puternice de multitasking în cazul în care anumite aplicații o cer.

Implementarea mecanismului de multitasking necesită, cel puțin, posibilitatea de a lansa, opri, comunica și sincroniza activitatea unor task-uri, precum și de a declara și utiliza relații de paternitate între task-uri ; sistemul de operare **MIX** rezolvă toate aceste cerințe.

3.2.1. Lansarea și oprirea task-urilor

Sistemul de operare **MIX** asigură serviciile sistem ce permit lansarea în execuție și oprirea unor task-uri independente. Task-urile pot fi activate de către operator sau de către alte task-uri. Activarea poate fi condiționată de apariția unor evenimente externe sau se poate produce imediat. Ea poate de asemenea avea loc la intervale de timp precizate (sincronizări, planificări, replanificări bazate pe timp). Task-urile pot fi de asemeni oprite de către operator sau de alte task-uri.

Toate aceste servicii sistem permit un control efectiv al multiprogramării între task-uri. Cu toate acestea, ele nu constituie un suport unic pentru multitasking ; acesta urmează a fi implementat de către utilizatorul însuși.

De exemplu, în figura 3.3, dacă taskul **A** solicită lansarea în execuție a task-ului **B**, Monitorul va încerca să rezolve această cerere și, în caz pozitiv, task-ul **B** va concura pentru acapararea resurselor sistem în paralel cu task-ul **A**. Task-ul **A** a creat un nou nivel de multiprogramare în sistem. Cu toate acestea, Monitorul nu poate informa task-ul **A** dacă acesta se execută în mod corect sau dacă task-ul **B** a activat nivelele inferioare **C** și **D**. Mai mult, task-ul **B** nici nu poate ști care din task-urile existente în sistem l-a lansat în execuție. Transmiterea de date între nivele, identificarea relațiilor dintre task-uri și comunicarea între task-uri, sînt cerințe ale multitasking-ului, rezolvate în cadrul sistemului **MIX** de către serviciile sistem de comunicare între task-uri.

3.2.2. Comunicarea și sincronizarea între task-uri

Tehnicile de comunicare și sincronizare utilizate în cadrul sistemului **MIX** sînt următoarele :

- indicatori de evenimente globali, comuni și de grup ;
- cutii poștale ;
- zone partajate ;
- conectarea și transmiterea de stare între task-uri care au stabilite relații de paternitate ;
- sincronizarea prin bitul de stopare ;
- fișiere partajate ;
- terminale virtuale (numai pe sistemul **MIX-PLUS**).

În toate cazurile, task-urile implicate în procesul de comunicare și sincronizare trebuie să efectueze verificări periodice, să se autosuspende sau să aștepte completarea unor evenimente asociate acestor tehnici. În general, aceste acțiuni sînt îndeplinite prin utilizarea mecanismului indicatorilor de evenimente, descris în detaliu în paragrafele următoare.

Sistemul de operare **MIX** oferă posibilități deosebite utilizatorilor în implementarea mecanismului de multitasking (acesta fiind egal sau mai puternic decît mecanisme similare în sisteme mult mai costisitoare), asigurînd toate primitivele necesare implementării unor scheme proprii de mutitasking, de orice grad de complexitate.

3.2.3. Relaţii de paternitate între tesk-uri

Prin utilizarea relaţiilor de paternitate între task-uri se pot stabili şi controla, pentru aplicaţiile în timp real, relaţii complexe între task-uri „tată“ (ce creează aceste relaţii) şi „fii“ (create la nităiativa task-urilor „tată“).

Un task „tată“ este acel task care lansează în execuţie şi se conectează la un alt task, numit task „fiu“. La terminarea execuţiei task-ului „fiu“, task-ul „tată“ primeşte informaţii de stare privind execuţia task-ului „fiu“ (terminare cu succes sau apariţia unor condiţii de eroare specifice).

O aplicaţie importantă a relaţiilor de paternitate între task-uri o reprezintă, în sistemul de operare **MIX-PLUS**, subsistemul de prelucrări pe loturi (batch). În acest caz, relaţiile şi parametrii asociaţi acestora pot fi stabiliţi on-line, iar controlul execuţiei uneia sau mai multor lucrări batch are loc off-line.

Sincronizarea execuţiei între task-urile „fii“ şi „tată“ are loc prin utilizarea relaţiilor de paternitate între task-uri :

— iniţierea execuţiei (activarea) unui task „fiu“ ;

— conectarea/deconectarea la/de la un task „fiu“ ;

— emiterea de informaţii către un task „fiu“, cu activarea şi, eventual, conectarea la acesta ;

— trecerea informaţiilor de conectare de la un task „tată“ la un task „fiu“, pentru crearea unor ierarhii de paternitate (subtasking) ;

— emiterea de stare către un task „fiu“ conectat sau, de la aceasta, către un task „tată“ ;

— terminarea (normală sau anormală) a execuţiei unui task „fiu“ cu transmiterea unor informaţii de stare către task-ul „tată“ ;

Un task „tată“ poate activa şi se poate conecta la mai multe task-uri „fii“ ; de asemeni, un task „tată“ se poate conecta multiplu la acelaşi task „fiu“. Un task „fiu“ poate fi conectat la mai multe task-uri „tată“.

Pentru realizarea relaţiilor de paternitate între task-uri se pot utiliza două tipuri de directive sistem :

— de activare şi creare a conexiunii între task-urile „tată“ şi „fiu“ (spawning) ;

— de transfer la un nivel inferior a conexiunii între un task „tată“ şi „fiu“ (chaining).

3.3. Task-uri.

3.3.1. Conceptul de task

Task-ul reprezintă unitatea de lucru a familiei de sisteme de operare **MIX**. Un *task* constă dintr-unul sau mai multe programe scrise într-un *limbaj sursă*, cum ar fi limbajul de macroasamblare (**MAC**) sau **FORTRAN** (**FOR**), asamblate sau compilate într-un format obiect şi transformate în imagine executabilă de către Editorul de legături (**TKB**).

Editorul de legături, pe lîngă funcțiile normale de editare, inițializează și atributele de bază ale task-ului, ce determină cererile de resurse și relațiile cu alte task-uri în sistem.

După editarea de legături, un task poate fi instalat în sistem și executat. Instalarea în sistem implică :

— înregistrarea atributelor task-ului în sistem ;
— alocarea resurselor necesare ;
— aducerea task-ului în memorie de pe un suport extern (dacă s-a solicitat fixare) ;
— plasarea task-ului în competiție cu alte task-uri active.

Numărul de task-uri instalate la un moment dat în sistem este limitat de memoria fizică disponibilă.

Fiecare task poate fi executat independent sau în conjuncție cu alte task-uri din sistem.

Task-urile pot fi lansate în execuție din următoarele moduri :

— de către utilizator, de la un terminal ;
— de către un alt task aflat în execuție ;
— pe baza planificării dependente de timp.

Planificarea se poate face imediat sau într-un interval de timp viitor, cu posibilități opționale de replanificare la intervale de timp periodice.

3.3.2. Atributele unui task

Atributele unui task sînt definite de cel care creează task-ul, fie prin specificarea unor opțiuni, fie prin utilizarea unor valori standard implicite. Un task posedă următoarele atribute :

— are un nume ;
— are o anumită dimensiune ;
— este privilegiat sau neprivilegiat ;
— utilizează sau nu instrucțiuni de virgulă mobilă ;
— necesită spațiu pentru stivă ;
— poate fi sau nu fixat în memorie ;
— se poate termina anormal sau nu ;
— poate include un sistem de depanare ;
— poate fi trasat ;
— este format din module concatenate sau nu ;
— poate fi evacuabil ;
— poate avea sau nu o structură multi-utilizator ;
— poate avea referințe la biblioteci partajate ;
— are o anumită prioritate ;
— se poate încărca într-o anumită partiție.

Atributele unui task pot fi precizate atît *static*, în momentul creării acestuia, cît și *dinamic*, la instalare sau în timpul execuției (prin utilizarea anumitor servicii sistem). Schematic, definirea acestor atribute se poate face în patru faze :

a) *Faza 1 — Trecerea de la sursă la obiect*

În această fază, un atribut important este statutul de privilegiat/neprivilegiat, întrucît anumite drepturi de acces acordate task-urilor privilegiate (de ex., accesul la pagina externă) sînt interzise task-urilor neprivilegiate.

De asemenea, caracteristicile asociate diferitelor secțiuni sursă (**.PSECT**) sînt importate în construirea imaginii finale a task-ului utilizator (structură mono sau multi-utilizator, etc.).

b) *Faza 2 — Trecerea de la obiect la imagine executabilă*

În această fază, se precizează următoarele atribute :

— *Numele :*

Numele poate fi specificat la editarea de legături sau instalare. În cazul în care nu se specifică, se utilizează implicit numele fișierului imagine executabilă.

— Dimensiunea :

Lungimea maximă a imaginii executabile, rotunjită la multiplul superior de 32 cuvinte.

— *Privilegii :*

Task-urile privilegiate necesită o relocare specială în memorie. Relocarea e verificată în faza 3.

— *Opțiunea de virgulă mobilă :*

Task-urile care intenționează să utilizeze instrucțiunile de virgulă mobilă trebuie să specifice acest lucru din faza 2. Neindicarea acestui lucru poate duce la rezultate eronate sau erori sistem.

— *Cererea de spațiu pentru stivă :*

Alocarea de spațiu se face în faza 2.

— *Opțiunea de fixare :*

Un task poate fi fixat sau nu în memorie.

— *Opțiunea de terminare anormală :*

Un task poate fi sau nu terminat anormal. Utilizarea opțiunii previne terminarea anormală a unor task-uri, în special componente sistem.

— *Opțiunea de depanare :*

În cazul selectării acesteia, sistemul de depanare **ODT** este încărcat odată cu task-ul apelant.

— *Opțiunea de trasare :*

Un task poate fi trasat prin setarea bitumului **T** în cuvîntul de Stare program (**PS**) inițial al task-ului.

— *Acces la biblioteci partajate :*

Referințele la biblioteci partajate se rezolvă complet în faza 2 prin căutarea acestora în fișierele de simboluri asociate bibliotecilor și includerea unor specificatori de biblioteci în imagine task.

Și alte atribute pot fi precizate sau modificate în cadrul Editării de legături. În majoritatea cazurilor, acestea pot fi precizate și în mod implicit.

c) *Faza 3 — Instalarea unui task*

În aceast fază pot fi modificate atribute importante ale task-ului, cum ar fi :

— dimensiunea ;
— condiția de terminare anormală ;
— numele ;
— prioritatea ;
— partiția, etc.

3.3.3. Instalarea unui task

Instalarea este procedeul prin care un task este introdus în sistem. În cazul instalării unui task, (prin emiterea unei comenzi **MCL/DCL INS**tall de la un terminal), sistemul de operare **MIX** înregistrează un număr de parametri și atribute atașate task-ului într-o tabelă sistem (**TCB** — Task Control Block = = Bloc de control al task-ului), pe care o inserează în lista generală a task-urilor din sistem (**STD**— System Task Directory). Dintre parametrii înregistrați, reamintim :

— numele task-ului ;
— lungimea acestuia ;
— numele partiției în care task-ul urmează a fi lansat în execuție ;
— atribute și privilegii.

În cazul sistemelor de operare **MIX/MIX-PLUS**, rezidente disc, instalarea semnifică de obicei numai înregistrarea parametrilor task-ului (crearea și inițializarea **TCB**) și nu încărcarea în memorie (aceasta are loc imediat după lansarea efectivă în execuție a task-ului).

În sistemul de operare **MIX-RT**, instalarea este însoțită și de fixarea (încărcarea task-ului de pe suport extern) în memorie. Opțiunea similară există și sub sistemul de operare **MIX.**

Fixarea permite o inițiere și o replanificare rapidă a task-urilor de către sistem, atribute esențiale în aplicațiile de timp real sau conducere de procese.

Eliminarea unui task din sistem are loc explicit prin execuția unei comenzi operator **MCL/DCL REMOVE** (emisă de la un terminal/utilizator privilegiat.).

3.3.4. Task-uri privilegiate și neprivilegiate

Un task privilegiat este un task cu drepturi de acces speciale la memorie și la dispozitivele de intrare/ieșire. Un task privilegiat are acces la partiția proprie, la zona Monitor și la pagina de intrare/ieșire și poate trece peste protecțiile software de acces, verificate la nivelul rutinelor sistem.

Un task neprivilegiat are acces numai la partiția propie și la dispozitivele de intrare/ieşire cu care interacționează.

Într-un sistem fără relocare, un task poate avea acces la întreaga memorie fizică, neexistînd distincție între task-urile neprivilegiate și cele privilegiate. În acest caz, se solicită expres utilizatorilor sistemului să respecte regulile de acces la anumite zone de memorie și să facă deosebirea între task-urile privilegiate și neprivilegiate.

Într-un sistem cu relocare, un task poate avea acces numai la memoria specifică aflată în posesia task-ului, ceea ce duce la accentuarea diferențelor între task-urile privilegiate și neprivilegiate.

Dacă task-urile privilegiate conțin referințe la subrutinele sau structurile de date Monitor, ele vor fi relocate în spațiul virtual, în continuarea zonei Monitor. Dacă Monitorul este mai mic de 16 kcuvinte, dimensiunea unui astfel de task poate fi de maximum 12 kcuvinte. Dacă Monitorul atinge 20 kcuvintè, dimensiunea unui task privilegiat poate fi de maximum 8 kcuvinte.

Task-urile privilegiate pot fi potențial periculoase pentru buna funcționare a sistemului de operare, putînd genera erori ce conduc la coruperea structurilor de date Monitor sau la execuția eronată a unor rutine Monitor. Din aceste motive, se recomandă o atenție sporită în procesul de dezvoltare, testare și execuție a task-urilor privilegiate utilizator, precum și analizarea corectă a cauzelor căderilor sistem *(crash)*.

Exemple de task-uri privilegiate în sistemul de operare **MIX** sînt : Procesorul de comenzi operator, Funcțiile operator extinse, Procesorul de comenzi indirecte, etc.

Exemple de task-uri neprivilegiate în sistemul de operare **MIX** : Macroasablorul, Editorul de legături, Programele utilitare, etc.

3.3.5. Contextul unui task

Ca unitate de lucru a sistemului de operare **MIX**, fiecare task posedă un context hardware și software.

Contextul hardware constă din valorile încărcate în registrele unității centrale atunci cînt task-ul este planificat pentru execuție :

— registrele generale **(R0 ÷ R7)** ;
— stare program **(PS)** ;
— registre de relocare a spațiului virtual ;
— registre de lucru ale altor procesoare (virgulă mobilă, etc.).

Contextul hardware al minicalculatorului este unic și este utilizat la un moment dat de un sinur task. Pe durata execuției task-ului acest context hardware este actualizat continuu. În momentul întreruperii execuției, ca urmare a apariției unui eveniment semnificativ, contextul hardware este salvat într-un context software, format din :

— tabela de control a task-ului **(TCB)** ;
— zone de reentranță task ;
— stivă task.

Salvarea contextului hardware în contextul software al task-ului curent aflat în execuție și încărcarea unui nou context hardware se numește *schimbare de context*.

Schimbarea de context are loc la fiecare planificare în execuție a unui task, de către planificatorul general al sistemului de operare **MIX**.

3.4. Evenimente

Evenimentele indică o schimbare a stării task-urilor sau terminarea unor activități ce privesc unul sau mai multe task-uri ce comunică între ele.

Evenimentele pot fi semnificative sau particulare, sincrone sau asincrone cu execuția unui task.

3.4.1. Tipuri de evenimente

Un *eveniment semnificativ* constituie o schimbare a stării generale a sistemului ceea ce duce la reevaluarea posibilităților de execuție a tuturor task-urilor din sistem.

În general, apariția unui eveniment semnificativ se datorează (direct sau indirect) execuției unei directive Monitor într-un task.

Evenimentele semnificative apar în următoarele situații :

— la completarea (terminarea) unei operații de intrare/ieșire ;

— în cazul comunicării între task-uri prin cutii poștale (execuția directivelor Monitor **SDAT$, SDRC$, SDRP$**) ;

— în cazul comunicării de referințe la regiuni partajate (execuția directivelor Monitor **SREF$, RREF$, RRST$**) ;

— la schimbarea priorității unui task (execuția directivei **ALTP$**) ;

— la epuizarea unor intervale de timp specificate (rezultînd din execuția directivei **MRKT$** sau unei cereri de replanificare) ;

— la forțarea apariției unui eveniment semnificativ (execuția directivei **DECL$S**) ;

— în cazul expirării intervalelor de timp de planificare ale algoritmului de planificare ciclică sistem *(round-robin)* ;

— la terminarea (normală sau anormală) a execuției unui task ;

— la execuția directivelor Monitor **EXIT$S, EXST$** sau **EMST$**.

Pentru așteptarea apariției unui eveniment semnificativ, se utilizează **WSIG$S**. Apariția unui eveniment semnificativ nu afectează numai planificarea execuției task-urilor în sistem, ci și coordonarea activității interne a task-urilor sau comunicarea între ele.

De exemplu, un task poate emite o directivă Monitor, asociind un indicator de eveniment unui eveniment semnificativ oarecare. La apariția evenimentului semnificativ, Monitorul setează indicatorul de eveniment specificat. Prin testarea stării indicatorului, task-ul poate determina dacă evenimentul semnificativ a apărut sau nu.

Evenimentele particulare coordonează numai activitatea internă a unor task-uri.

3.4.2. Indicatori de evenimente

Indicatorii de evenimente (event flags) reprezintă mijloacele prin care task-urile recunosc apariția unor evenimente ; unele evenimente specifice unui task pot fi recunoscute și cu ajutorul întreruperilor asincrone (ASI_2.

Fiecărui eveniment îi este asociat un bit (numit indicator), care poate fi setat (pus pe unu) sau șters (pus pe zero) în funcție de apariția evenimentului asociat indicatorului.

Indicatorii de evenimente constituie principala tehnică utilizată în sistemul de operare **MIX** pentru sincronizarea activității în cadrul unui task sau între task-uri (vezi cap. 4).

Pentru identificarea evenimentelor între ele, task-urile au la dispoziție nouăzeci și șase (96) indicatori de evenimente.

Acești indicatori de evenimente se împart în trei grupe :

— indicatori locali unui task ;

— indicatori comuni tuturor task-urilor din sistem (indicatori globali);
— indicatori comuni unor grupuri de task-uri (indicatori globali de grup).

Primii 32 de indicatori (1 ÷ 32) sînt locali pentru fiecare task și se găsesc în blocul de descriere al task-ului (TCB_2). Indicatorii pot fi setați sau șterși numai ca urmare a execuției task-ului proprietar.

Următorii 32 de indicatori (33 ÷ 64) sînt comuni tuturor task-urilor și se găsesc într-o zonă sistem. Indicatorii pot fi setați sau șterși în urma execuției oricărui task în sistem.

Ultimii 32 de indicatori (65 ÷ 96) sînt utilizați în aplicațiile ce necesită sincronizare pe grupuri de task-urile. Acești indicatori pot fi utilizați numai de acele task-uri ale căror coduri de identificare (**UIC**) conțin coduri de grup identice cu codurile de grup ale evenimentelor globale (putînd exista maximum 255 de seturi de cîte 32 de indicatori de grup, corespunzînd codurile de grup octale 1 la 377).

Ultimii 8 indicatori din grupa indicatorilor locali (25 ÷ 32) și indicatorilor comuni (57 ÷ 64), sînt rezervați sistemului de operare **MIX**, pentru o utilizare implicită, în componentele sistem.

Sistemul de operare **MIX** asigură o serie de servicii relative la evenimente și indicatori de evenimente, capabile să îndeplinească următoarele funcții asupra unor indicatori de evenimente locali și globali — comuni și de grup:

— a) servicii relative la indicatorii locali și comuni:
  — ștergere indicator de eveniment (**CLEF$**);
  — setare indicator de eveniment (**SETF$**);
  — citire indicator de eveniment (**RDEF$**);
  — citire indicatori de evenimente (**RDAF$**);
  — așteptare pe indicator de eveniment (**WTSE$**);
  — așteptare multiplă pe indicatori de evenimente (**WTLO$**);
  — stopare pe indicator de eveniment (**STSE$**);
  — stopare multiplă pe indicatori de evenimente (**STLO$**).

b) servicii relative la indicatorii globali de grup:
  — creare grup de indicatori de evenimente globali (**CRGF$**);
  — eliminare grup de indicatori de evenimente globali (**ELGF$**);
  — deblocare acces la grup de indicatori de evenimente globali (**ULGF$**);
  — citire extinsă a indicatorilor de evenimente (**RDXF$**).

Indicatorii locali de evenimente sînt utilizați în sincronizarea activității interne în cadrul unui task.

Indicatorii comuni de evenimente pot fi utilizați atît pentru sincronizarea activității interne a unui task, cît mai ales pentru sincronizarea activității între task-uri.

Utilizarea eronată a indicatorilor comuni de evenimente (setări și ștergeri multiple, nesincronizări) poate duce la anomalii utilizator sau sistem greu de depistat, iar în cazul unor task-uri care comunică între ele la interblocare *(deadlock)*.

Programele utilizator pot fi scrise și fără o utilizare explicită a indicatorilor de evenimente locali sau comuni, întrucît directivele sistem pot acționa asupra lor și implicit. Utilizarea implicită a indicatorilor de evenimente asi-

gură o reducere substanțială a cazurilor de eroare datorate unor setări și ștergeri multiple sau nesincronizări. Directivele **SDAT$, MRKT$, QIO$ și QIOW$** pot altera implicit un indicator de eveniment specificat.

Fiecărui indicator de eveniment îi corespunde un *număr de indicator de eveniment* (**EFN**), utilizat la nivelul task-ului utilizator pentru identificarea fiecărui eveniment.

De cele mai multe ori, poziționarea unui indicator de eveniment nu este forțată de o directivă sistem, ci de apariția unei întreruperi (cum ar fi de exemplu, terminarea unei operații de intrare/ieșire). Cu toate acestea, poziționările de indicatori nu provoacă întreruperea task-ului aflat în execuție, ci sincronizarea execuției acestuia.

De exemplu, în cazul în care un task se află în execuție și apare un eveniment semnificativ, task-ul poate fi plasat în una din următoarele stări :

— poate fi suspendat, întrucît evenimentul era așteptat de un task gata de a fi lansat în execuție, de prioritate mai mare. Task-ul suspendat este plasat în starea gata de a fi lansat în execuție ;

— își continuă execuția ; dacă evenimentul semnificativ era adresat task-ului activ, acesta este relansat din punctul de suspendare temporară, cu o stare diferită de cea anterioară : în acest moment are un indicator de eveniment poziționat. Atîta vreme cît task-ul nu testează în mod explicit această schimbare de stare, execuția sa va continua ; task-ul va descoperi poziționarea evenimentului fie printr-un test explicit (apel directivă **Monitor**), fie forțînd o așteptare pe indicatorul de eveniment corespunzător.

În cazul în care un task dorește să-și sincronizeze execuția pe apariția unui eveniment oarecare (în special, terminarea unor operații de intrare/ieșire — **QIO$|QIOW$** sau epuizarea unor intervale de timp — **MRKT$**), acesta emite o directivă de autoblocare (așteptare) pe unul (**WTSE$**), sau mai multe (**WTLO$**) evenimente). Dacă acest(ea) sînt deja poziționat(e), condiția de blocare este satisfăcută imediat și se redă controlul task-ului apelant ; dacă nu, task-ul apelat va fi blocat pînă la apariția evenimentului specificat.

O alternativă pentru task a condiției de autoblocare (așteptare) este cea de stopare, prin emiterea unei afective de stopare pe unul (**STSE$**) sau mai multe evenimente (**STLO$**). În acest caz, în loc de blocarea task-ului în memorie, se forțează stoparea (și implicit evacuarea sa din memorie).

Un task blocat concură la alocarea resursei memorie cu prioritatea sa de execuție ; un task stopat are prioritatea zero (0).

3.5. Planificarea task-urilor

3.5.1. Conceptul de planificare

Planificarea task-urilor în sistemul de operare **MIX** reprezintă metoda de distribuire a resurselor sistem între task-urile active, rezidente în memorie, pentru realizarea următoarelor scopuri :

— utilizarea eficientă a resurselor sistem ;

— răspuns rapid la cererile de timp real.

În mod tradițional, planificarea înseamnă selectarea următorului task căruia i se pasează controlul unității centrale. În sistemele cu mai multe uni-

tăți centrale ce lucrează în paralel (**MIX-PLUS**), mecanismul de planificare trebuie să țină cont și de această situație. De aceea, planificarea trebuie văzută ca un proces în trei faze :

— alocarea resurselor particulare cerute explicit de task-urile din sistem ;

— selectarea cererilor de resurse partajate (memorie, dispozitive de intrare/ieșire comune, etc.) ;

— selectarea task-urilor pentru controlul unității centrale.

Planificarea nu constituie însă un prerogativ al Monitorului **MIX** ; task-urile să poată influența de asemenea deciziile de planificare ale Monitorului.

În cazul sistemului de operare **MIX**, la baza algoritmului de planificare implementat stă prioritatea task-urilor.

În general, la apariția unor cereri multiple pentru o resursă sistem, e necesară o disciplină de servire pentru ca una din cereri să poată căpăta controlul resursei. Dacă resursa constă din mai multe componente diferite, unele inactive și altele active, atunci partajarea resurselor disponibile și paralelismul constituie metodele de creștere a gradului de utilizare a resursei.

Unitatea centrală constituie o resursă cheie, dar nu și dispozitivele de intrare/ieșire, întrucît aceastea pot lucra în paralel cu unitatea centrală și pot servi cereri independente. Multiprogramarea constituie o metodă de creștere a utilizării resurselor sistem, prin multiplexarea cererilor de utilizare a acestora.

Problema pusă mecanismului de planificare este cea a selectării cererilor pentru resursele sistemului. Soluția aleasă depinde de obiectivul urmărit.

Pentru aplicațiile de timp real, cerințele de resurse sînt absolute, deseori sincronizate la intervale specifice de timp ; nerezolvarea lor poate avea rezultate catastrofale.

Pentru utilizatorii ce dezvoltă programe, de la pupitrul unor terminale, răspunsul sistemului trebuie să se înscrie în limite rezonabile de timp.

Întrucît un parametru deosebit de important în sistemele de timp real îl constituie eficiența de comutare a task-urilor, nu au fost alese alternative cu aplicabilitate generală costisitoare, ci o metodă de planificare rapidă, simplă și eficientă.

În sistemul de operare **MIX**, utilizatorii comunică sistemului parametrii de planificare prin specificarea :

— unei priorități ;

— unor cereri de planificare explicite.

3.5.2. Variabilele planificatorului de task-uri

Planificatorul de task-uri al sistemului de operare **MIX** își bazează funcțiile de planificare pe următoarele variabile :

— prioritatea task-urilor ;

— apariția unor evenimente semnificative și rezultatul tranzițiilor de stare ale task-urilor ;

— expirarea unor intervale de timp alocate unor task-uri obișnuite (cum ar fi depășirea unei cuante de timp) ;

— cereri de planificare explicite.

3.5.3. Priorităţi

Task-urile active concură pentru acapararea resurselor sistem pe baza priorităţii task-urilor şi a disponibilităţii resurselor.

În scopul unei mai bune planificări, în sistemul de operare **MIX** se definesc *250 niveluri distincte de prioritate software*. Priorităţile acoperă o gamă numerică de la 1 la 250 (zecimal), în care 250 reprezintă prioritatea software cea mai mare.

Sistemul de operare **MIX** nu alocă priorităţi speciale task-urilor, indiferent de tipul acestora. Sarcina alocării de priorităţi rămîne în seama utilizatorilor, operatorul sau a inginerului de sistem ce proiectează aplicaţia respectivă. În general se alocă priorităţile $1 \div 150$ unor task-uri obişnuite, iar cele din gama $151 \div 250$ unor task-uri critice în timp.

Task-urile critice în timp sînt reprezentate de task-urile de timp real, iar task-urile obişnuite sînt task-urile de dezvoltare de programe.

Task-urile critice în timp sînt planificate numai pe baza priorităţii ataşate acestora ; task-ul cu prioritatea cea mai mare, căruia i-au fost alocate toate resursele solicitate, capătă controlul unităţii centrale.

Task-urile de dezvoltare de progres pot fi planificate fie pe bază de priorităţi, fie utilizînd algoritmi speciali de planificare care să asigure o suprapunere maximă între activităţile de execuţie şi cele de intrare/ieşire.

Prioritatea unui task se poate specifica *static* la editarea de legături (opţiunea **PRI**) şi *dinamic* de către operatorul sistem (prin comenzile de instalare (**INS**), lansare în execuţie (**RUN**) sau modificare prioritate (**ALTER**)) şi utilizator (prin directivă Monitor **ALTP$**). O astfel de planificare se numeşte planificare internă a execuţiei task-urilor.

Diagrama de planificare a task-urilor după priorităţi este prezentată în figura 3.4.

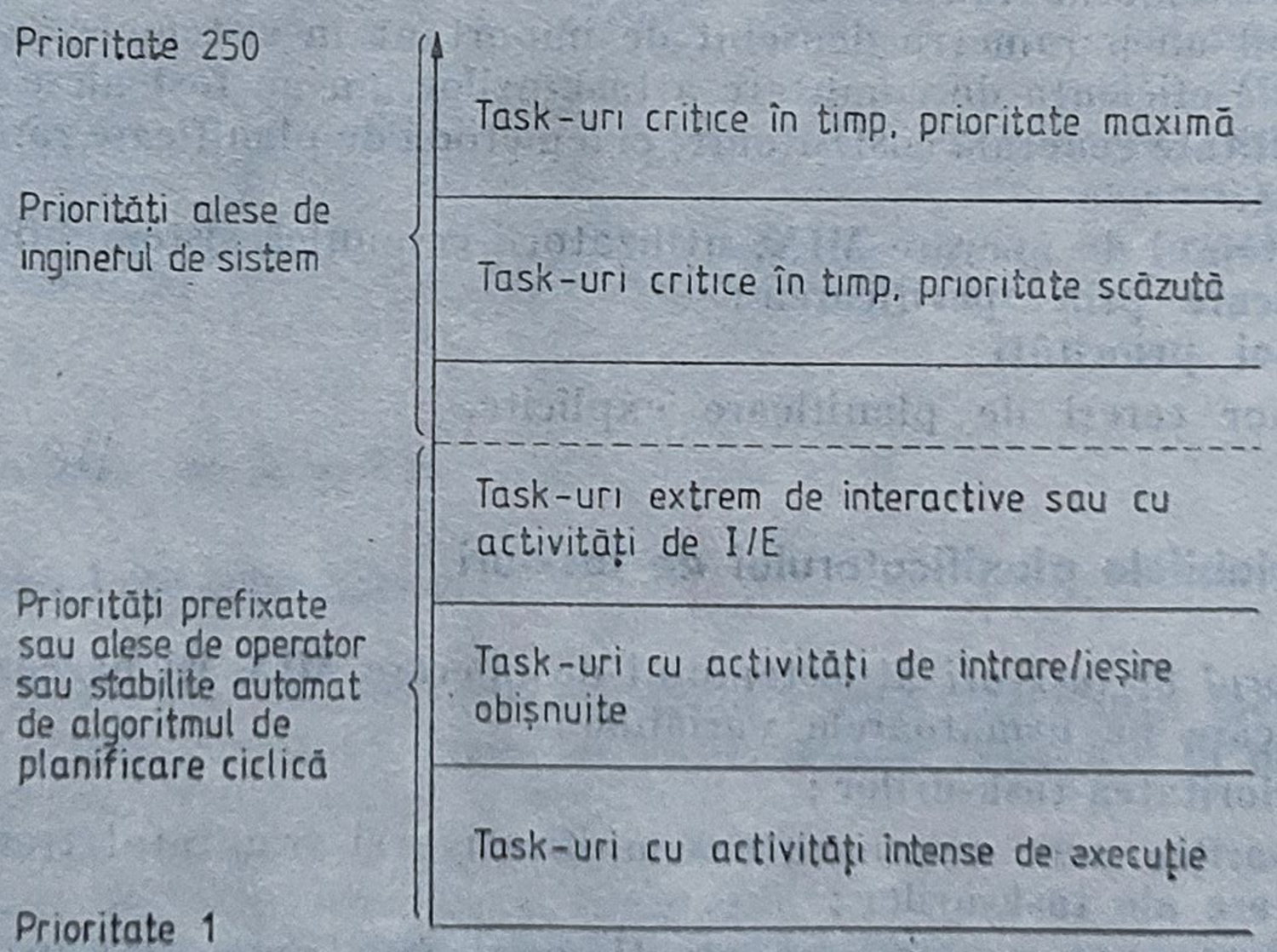


Fig. 3.4. Diagrama de planificare a task-urilor după priorităţi în sistemul de operare MIX

3.5.4. Evenimente semnificative

Evenimentele semnificative sînt apariţii ce duc la schimbarea stării unuia sau mai multor task-uri în sistem. Schimbarea stării curente este reflectată de extragerea tabelei de descriere a task-ului (**TCB**) dintr-o listă de stare şi inserarea acesteia în lista stării curente.

Un task aflat în execuţie poate genera un eveniment semnificativ prin autoblocare sau poate forţa un eveniment semnificativ pentru alt task. Mai mult, anumite componente sistem, cum ar fi tratarea timpului sistem, duc la apariţia unor evenimente semnificative. Indiferent de sursă, toate evenimentele semnificative sînt raportate planificatorului sistemului de operare.

Evenimentele semnificative pot fi sincrone cu un task aflat în execuţie (de exemplu, o cerere de autoblocare) sau asincrone (de exemplu, terminarea unui transfer de intrare/ieşire).

Indiferent de eveniment, task-ul selectat de planificator pentru a fi lansat în execuţie este întotdeauna task-ul de cea mai mare prioritate din lista de task-uri active.

3.5.5. Tranziţiile de stare ale unui task

Un task este recunoscut de către sistem numai după instalare. El este considerat a fi în hibernare pînă în momentul în care un alt task aflat în execuţie sau o comandă emisă de la un terminal solicită sistemului activarea acestuia.

Lansarea în execuţie are loc prin comanda operator **MCL/DCL RUN** sau prin utilizarea unei *directive sistem* (**RQST$, RUN$, SPWN$, RPOI$, SDRC$, VSRC$** sau **SDRP$**).

Starea unui task reprezintă condiţia acestuia la un moment dat. De exemplu, un task poate fi în hibernare sau în aşteptarea terminării unui eveniment. Stările posibile ale unui task se exclud reciproc.

Un task se poate găsi deci în una din următoarele două stări de bază :

a) *În hibernare*. Un task se găseşte în această stare imediat după instalare (i s-a alocat o intrare în **STD** şi, eventual, a fost încărcat în memorie, dar nu a fost lansat în execuţie) ;

b) *Activ*. Un task activ este un task instalat, căruia i s-a cerut lansarea în execuţie. El rămîne în această stare pînă în momentul terminării, normale sau anormale, cînd revine în starea de hibernare. Un task „activ" este eligibil pentru planificarea în execuţie, pe cînd un task „în hibernare" nu este.

Un *task activ* se poate găsi în una din următoarele *trei substări : gata de a fi lansat în execuţie, blocat* şi *stopat*.

— *Gata de a fi lansat în execuţie :*

Un astfel de tash, aflat în competiţie cu alte task-uri în funcţie de prioritate, concură pentru acapararea Unităţii centrale (UC). Task-ul cu prioritatea cea mai mare acaparează Unitatea centrală şi devine *task-ul curent ;*

— *Blocat :*

Un astfel de task nu poate concura pentru acapararea Unităţii centrale, din lipsa unor resurse sistem sau din necesităţi de sincronizare (internă

sau externă). Prioritatea task-ului rămîne neschimbată, permiţînd acestuia să concure la acapararea resursei memorie internă;

— *Stopat:*

Un astfel de task nu îşi poate continua execuţia deoarece aşteaptă terminarea unui transfer de I/E, apariţia unor evenimente neaşteptate sau s-a autostopat. În această stare, prioritatea task-ului este 0 (zero), aceasta putînd fi evacuat (din memorie) de către alte task-uri, indiferent de prioritatea acestora. La apariţia unei întreruperi asincrone (**AST**), prioritatea task-ului este restaurată pe durata execuţiei rutinei de servire **AST**, după care revine la zero.

Starea unui task şi prioritatea ataşată acestuia constituie informaţiile de intrare pentru planificatorul de task-uri din sistemul de operare **MIX**.

Tranziţiile unui task de la o stare la alta sînt rezultanta evenimentelor semnificative raportate planificatorului.

Tranziţiile posibile pot fi:

a) din „*hibernare*“ în „*activ*“, ca urmare a:
   — execuţiei unei directive sistem **RUN$, RQST$, SPWN$, SDRC$, VSRC$, RPOI$** sau **SDRP$**;
   — execuţiei comenzii operator **MCL** sau **DCL RUN**.

b) din „*gata de a fi lansat în execuţie*“ în „*blocat*“, ca urmare a:
   — execuţiei directivei **SPND$S**;
   — autoblocării pe o condiţie de aşteptare nesatisfăcută;
   — evacuarea unui task din memorie.

c) din „*gata de a fi lansat în execuţie*“ în „*stopat*“, ca urmare a:
   — execuţiei directivei **STOP$S**;
   — execuţiei unei directive **RCST$, SDRP$, GCCI$**, sau **VRCS$** în condiţiile inexistenţei unor pachete de date în coada de recepţie a task-ului;
   — nesatisfacerii condiţiilor specificate de directiva **STLO$** sau **STSE$**;
   — unei cereri de introducere de date buferate, de la un terminal, pe durata acesteia; task-ul ce a specificat această cerere poate fi stopat numai dacă: nu se găseşte în tratarea unei întreruperi asincrone (**AST**), este evacuabil şi regiunea în care are loc transferul de I/E este evacuabilă.

d) din „*blocat*“ în „*gata de a fi lansat în execuţie*“, ca urmare a:
   — execuţiei directivei **RSUM$**, de către task-ul curent, cu specificarea task-ului „blocat“;
   — execuţiei comenzii operator **MCLESume** sau **DCL CONTINUE**;
   — satisfacerii unei condiţii de aşteptare (autoblocare);
   — reîncărcării în memorie a unui task evacuat.

e) din „*stopat*“ în „*gata de a fi lansat în execuţie*“, ca urmare a:
   — execuţiei directivei **USTP$** (efectivă numai în cazul în care task-ul a fost stopat printr-o directivă **STOP$, RCST$** sau **VRCS$**);
   — completării unui transfer de I/E pentru un task cu mai multe asemenea operaţii în paralel;
   — setării (poziţionării) unuia sau mai multor indicatori de evenimente, dacă task-ul era stopat pe unul din aceştia;

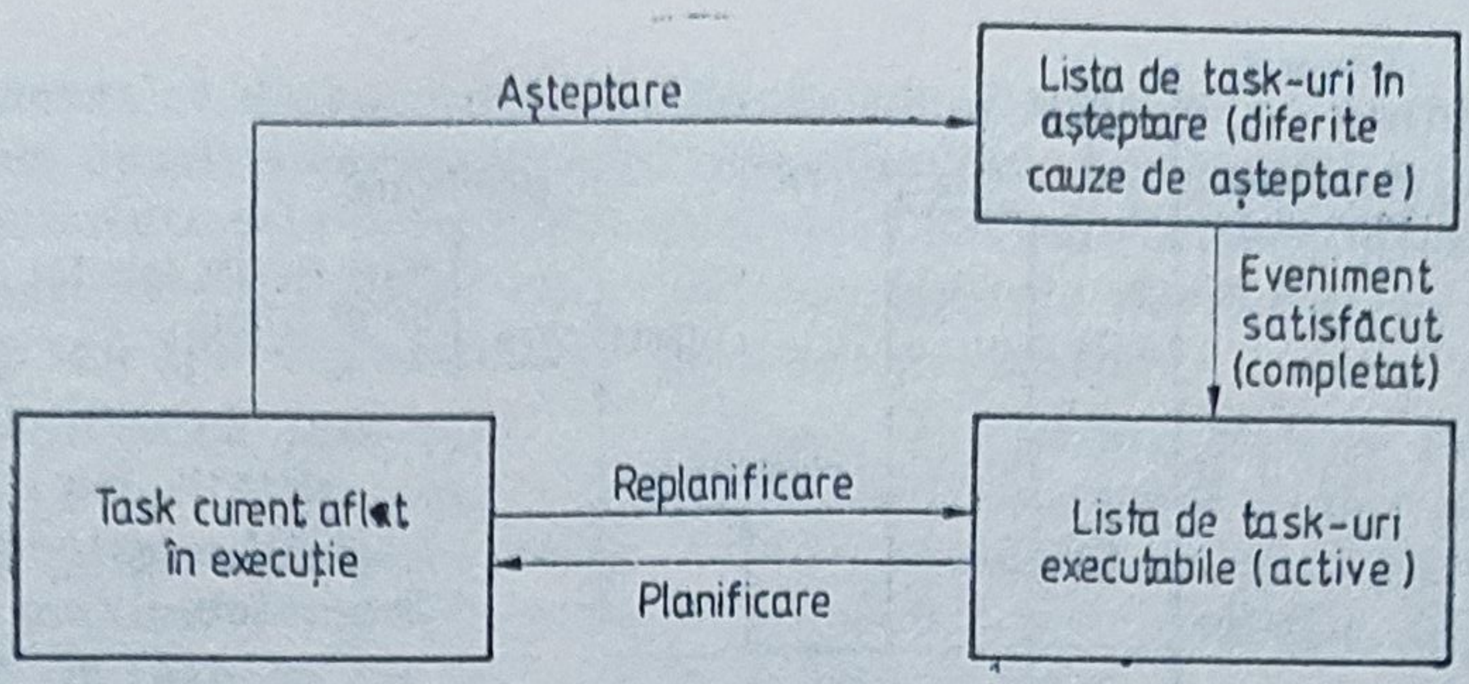


Fig. 3.5. Ciclul de tranziții între stările unui tas în listele sistemului

— execuției comenzii operator **MCL UNSTOP** sau **DCL START**;
— completării unei introduceri de date de la un terminal, pentru un task evacuabil.

f) din „*activ*“ în „*hibernare*“, ca urmare a :
— execuției unei directive **EXIT$S, EXIF$, RCVX$** sau **VRCX$**;
— execuției unei directive **RREF$** sau **GCCI$** (ce a specificat opțiunea de terminare execuție task);
— execuției unei directive **ABRT$**;
— execuției unei comenzi operator **MCL** sau **DCL ABORT**;
— apariția unei întreruperi software sincrone **(SST)** pentru care task-ul nu a specificat o rutină de tratare corespunzătoare.

g) din „*blocat*“ în „*stopat*“, ca urmare a :
— inițierii de către task a unei I/E buferate, într-o rutină de tratare **AST**, urmată de ieșirea din starea **AST**.

h) din „*stopat*“ în „*blocat*“, ca urmare a :
— completării (terminării) tuturor I/E buferate, așteptate de către task.

Ciclul tranzițiilor de stare ale unui task este prezentat în figura 3.5.

3.5.6. Expirarea unor intervale de timp

În cazul existenței unui număr mare de task-uri rezidente, de priorități egale sau apropiate, planificatorul va avea tendința să aloce mai des resursa timp Unitate centrală task-urilor ce apar în primele poziții ale listei de task-uri active.

Pentru a preveni această situație, sistemul de operare **MIX** implementează un mecanism de planificare suplimentar, selectabil la generare, numit *planificare ciclică (round-robin)*. Fiecărui task, indiferent de prioritatea sa (în gama de priorități selectată la generare sau dinamic, de operator), i se alocă o cuantă a timpului de execuție, ce asigură :
— un interval de timp minim garantat în care task-ul se execută fără a i se schimba poziția în listele sistem ;
— un interval de rotație pentru task-urile cuprinse într-o anumită gamă de priorități.

Parametrii algoritmului de planificare ciclică (gama de priorități și intervalul de timp minim) pot fi modificați static, la generare, sau dinamic,

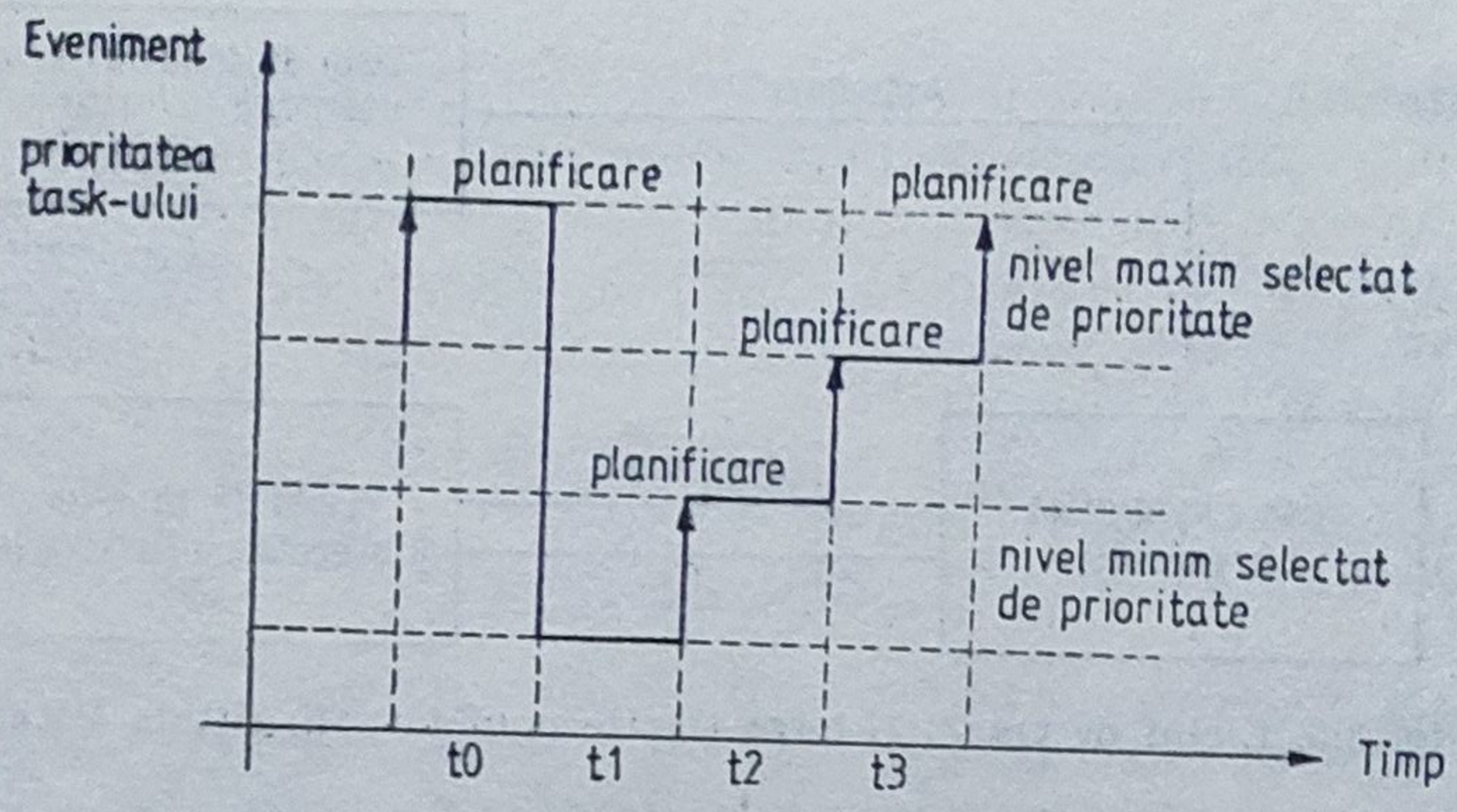


Fig. 3.6. Algoritmul de planificare ciclică a task-urilor din sistem

de către utilizatorii privilegiați, prin folosirea comenzii **MCL/DCL** sau **VMR SET**.

Efectul general al planificării ciclice este prezentat în figura 3.6. La expirarea unei cuante de timp, prioritatea task-ului aflat în execuție scade, pînă la nivelul minim specificat, după care, crește incremental, pînă la nivelul maxim de prioritate specificat.

Planificarea explicită (externă) a execuției task-urilor constituie o funcție importantă a utilizatorului, realizată de către interfața operator-sistem (comanda operator **MCL/DCL RUN**) sau directivele Monitor **(RQST$, RUN$)**.

Acest tip de planificare lucrează în conjuncție cu planificarea internă a task-urilor active bazată pe priorități. Operatorul sistem poate include parametri de timp într-o comandă ce activează un task instalat, inactiv, solicitînd Monitorului să execute un task :

— la un moment de timp specificat față de momentul curent ;

— la un moment de timp specificat față de unitatea de sincronizare a ceasului sistem ;

— la un moment de timp absolut ;

— imediat.

Cererile de activare pot fi însoțite sau nu de replanificări periodice.

3.5.7. Planificarea unui task pentru execuție

Planificarea unui task pentru execuție (obținerea controlului unității centrale) presupune luarea unei decizii minime. Task-ul selectat se găsește întotdeauna în vîrful listei de task-uri active din sistem.

Deciziile de planificare reale sînt luate ca rezultat al apariției unor evenimente semnificative, ce cauzează tranziții de stare și fac task-urile executabile.

În cazul în care un task este suspendat de către un altul de prioritate mai mare, task-ul întrerupt este plasat în coada taskurilor active, pe poziție corespunzătoare priorității acestuia (cu excepția algoritmilor de planificare ciclică).

Intervalul dintre întreruperile de planificare este aleatoriu, fiind determinat de apariția evenimentelor semnificative. În practică, timpul mediu între evenimente este determinat de durata unei operații de intrare/ieșire tipice.

Task-urile critice în timp nu sînt afectate de expirarea unor intervale de timp sau apariția unor evenimente semnificative.

Planificarea task-urilor în sistemul de operare **MIX** se face în ordinea strictă a priorităților pentru task-urile de timp real și după o prioritate modificată pentru celelalte. Aceasta permite planificatorului obținerea unei suprapuneri maxime de activități de intrare/ieșire și de execuție, păstrînd caracteristicile de răspuns ale task-urilor critice în timp.

3.6. Evacuarea/reîncărcarea task-urilor

3.6.1. Algoritmi de evacuare

În sistemele de operare rezidente disc (**MIX/MIX-PLUS**) sau rezidente în memorie (**MIX-RT** satelit în rețele de calculatoare), poate apare deseori situația în care task-urile active, nefixate în memorie, nu pot concura pentru acaparrarea resursei Unitate Centrală deoarece partiția în care sînt instalate este ocupată de către alte task-uri (comunuri) din sistem.

Dacă partiția respectivă conține unul sau mai multe task-uri, de prioritate mai mică decît cea asociată noului task, și dacă acestea sînt evacuabile, Monitorul le va evacua din memorie pe disc (local — **MIX/MIX-PLUS** sau la distanță — **MIX/RT**) pentru a face suficient spațiu task-ului sau task-urilor de prioritate mai mare.

Acest proces, ce duce la creșterea eficienței de utilizare a sistemului poartă numele de *evacuare (checkpointing)*.

În mod similar, la terminarea execuției unuia sau mai multor task-uri, sau cînd acestea devin blocate/stopate, task-ul sau task-urile evacuate sînt reactivate și își continuă execuția din punctul în care au fost întrerupte (la evacuare). Acest proces se numește *reîncărcare (reloading)*.

Înainte de reîncărcarea altor task-uri, task-urile blocate sînt evacuate cu prioritatea lor de instalare ; task-urile stopate sînt evacuate cu prioritatea zero (0), prioritatea de instalare fiind restabilită după completarea condiției de stopare. În mod particular, task-urile rezidente în memorie sînt stopate pe durata introducerii de date de la terminal, putînd fi întotdeauna evacuate, indiferent de prioritatea acestora.

Posibilitatea de evacuare a unui task poate fi validată sau invalidată dinamic prin utilizarea unor directive Monitor (**ENCP$S**, respectiv **DSCP$S**). Utilizarea acestor directive este oportună atunci cînd se dorește protejarea unor porțiuni critice în timp, ce ar fi afectate de durata evacuării/reîncărcării în memorie.

Atributul de evacuare, asociat unui task, poate fi precizat atît static, la editarea de legături (opțiunile /**CP** sau /**AL**), cît și dinamic, la instalare (/**CKP**) sau la execuția task-ului (directivele **ENCP$S/DSCP$S**).

Privind evacuarea, apare o problemă specială în Monitor, în cazul în care mai multe task-uri de prioritate egală concură pentru alocarea memoriei în aceeași partiție. În mod normal, un task nu poate forța Monitorul să evacueze un alt task de prioritate egală sau mai mare. În schimb, unui task de prioritate egală sau mai mică nu i se permite accesul la resursa memoriei.

Pentru rezolvarea acestei situații, algoritmul general de planificare din sistemele de operare **MIX/MIX-PLUS** (rezidente disc) a fost completat cu un alt algoritm, de *swapping*, ce permite Monitorului evacuarea din memorie a task-urilor (evacuabile) de prioritate egală.

La lansarea în execuție a unui task Monitorul adaugă, la prioritatea normală de execuție a acestuia, o altă prioritate numită de „evacuare". În timpul execuției task-ului, Monitorul decrementează prioritatea de evacuare (ce poate avea chiar valori negative) la un interval de timp minim garantat. În momentul în care suma dintre prioritatea de evacuare (decrementată) și prioritatea de execuție a task-ului devine mai mică decît prioritatea altui task, ce concură pentru acapararea memoriei, Monitorul evacuează task-ul curent pentru a face loc task-ului ce concură la resursa memoriei. Monitorul plasează task-ul evacuat în lista task-urilor active, asociată partiției de memorie la care există concurență.

Prioritatea de evacuare afectează numai alocarea de spațiu în partiții. Ea nu afectează planificarea Unității Centrale, la baza acesteia rămînînd prioritatea de execuție a task-urilor.

Parametrii algoritmului de swapping (gama de priorități și intervalul de timp minim) pot fi modificați static la generare, sau dinamic, de către utilizatorii privilegiați, prin folosirea comenzii **MCL/DCL** sau **VMR SET**.

Un caz particular, în sistemul de operare **MIX-PLUS**, îl reprezintă evacuarea zonelor partajeaze (blocuri de comun, statice și dinamice, biblioteci de rutine, zone „pure" al task-urilor multi-utilizatore) la care se referă un task evacuabil. Aceste zone partajate pot fi evacuate implicit (la evacuarea tuturor task-urilor ce se referă) sau explicit (prin execuția directivei Monitor **CPCR$**).

Readucerea în memorie a unui task mapat la o zonă partajată (statică sau dinamică) forțează și încărcarea acestui bloc de comun în memorie.

În sistemul de operare **MIX-RT** (satelit în rețele de calculatoare), task-urile sînt evacuate/reîncărcate, prin linii de comunicație, pe suporturi disc aflate la distanță (pe sistemele gazdă-noduri centrale).

3.6.2. Zone de evacuare statice și dinamice

Pentru a putea fi evacuat, un task necesită un spațiu corespunzător pe disc (local sau la distanță), egal cu dimensiunea task-ului sau partiției în care se execută. După salvarea task-ului evacuat, în acest spațiu, un task de prioritate mai mare poate ocupa zona de memorie (partiția) eliberată.

Spațiul de evacuare poate fi localizat (specificat) fie în mod static, la editarea de legături (per task), fie în mod dinamic, în timpul execuției task-urilor (per sistem).

Static, la editarea de legături se poate solicita ca spațiul de evacuare să fie alocat în imaginea task (/AL). În acest caz, spațiile de evacuare asociate task-urilor sînt permanent prezente pe disc, indiferent dacă Monitorul soli-

cită sau nu evacuarea acestora din memorie. Poziția și dimensiunea zonei de evacuare din imaginea task pot fi schimbate numai printr-o nouă editare de legături, cu parametri diferiți. Acest tip de alocare se folosește, în sistemul de operare **MIX**, pentru acele task-uri sistem, critice în timp, cu număr mare de evacuări.

Alocarea dinamică a spațiului de evacuare (prin execuția comenzii operator privilegiate **ACS**), permite o utilizare mai bună a spațiului de pe disc. În loc să se rezerve spațiu pentru fiecare task evacuabil — uneori chiar nenecesar — se creează pe disc(uri) unul (sau mai multe) fișiere de evacuare, ce vor include toate task-urile evacuabile din sistem.

Dimensiunea acestor fișiere depinde de estimările spațiului de evacuare solicitat de către sistem la un moment dat. În acest caz nu este necesară declararea atributului de evacuare (**/CP**) la editarea de legături a task-urilor. Această caracteristică poate fi specificată dinamic, la instalare sau prin execuția unor directive Monitor.

La solicitarea de evacuare a unui astfel de task, Monitorul controlează dacă task-ul este evacuabil și dacă da, îl elimină din memorie, într-o anumită zonă din fișierul de evacuare. În cazul unei evacuări ulterioare, task-ul poate fi plasat în altă parte, în fișierul de evacuare, decît în cazul anterior.

4

Comunicarea și sincronizarea între task-uri

Familia de sisteme de operare **MIX** permite comunicarea între task-uri în scopul sincronizării execuției acestora, transmiterii de mesaje (informații) sau exploatării în comun a unor zone de date sau programe.

Tehnicile de comunicare și sincronizare utilizate sînt :

— indicatori globali de evenimente — comuni și de grup ;

— cutii poștale (emisie/recepție mesaje) ;

— zone partajate ;

— conectarea și transmiterea de stare între task-uri care au stabilit relații de paternitate ;

— sincronizarea prin bitul de stopare ;

— terminale virtuale (numai pe sistemul **MIX-PLUS**) ;

— fișiere partajate.

4.1. Indicatori globali de evenimente, comuni și de grup

Indicatorii globali de evenimente — comuni și de grup — reprezintă poziționări de stare ce indică completarea sau apariția unor evenimente ce privesc mai multe task-uri ce comunică între ele.

În aplicațiile ce necesită execuția simultană a mai multor task-uri, indicatorii globali de evenimente pot fi utilizați drept mijloc de comunicare între ele și de sincronizare a activității acestora (vezi subcapitolul 3.4.2).

În capitolul 28 se explicitează și se exemplifică utilizarea acestor facilități în programarea în limbajul **MACRO**.

4.2. Cutii poștale

O *cutie poștală* reprezintă o zonă tampon din memoria dinamică a sistemului de operare **MIX**, tratată ca un dispozitiv de intrare/ieșire orientat pe înregistrare.

Un task creează o cutie poștală, în care înscrie informații ce pot fi preluate sau modificate de alte task-uri din sistem. Cutiile poștale sînt utile pentru transferarea unor informații de stare, coduri de retur, mesaje, sau orice alt gen de informații.

Fiecărui task existent în sistem i se alocă o cutie poștală în care celelalte task-uri pot depune mesaje. Mesajele sînt emise către cutia poștală prin programarea unei directive Monitor **SDAT$**, **VSDA$**, **SDRC$**, **SDRP$** sau **VSRC$**. În acest moment se și creează o intrare în cutia poștală a task-ului destinatar. Sistemul introduce această intrare în coada de recepție a task-ului destinatar (ordonată FIFO și descrisă de locația **T.RCVL** din **TCB**), indiferent de starea task-ului (activ sau inactiv).

În momentul emiterii unui mesaj către o cutie poștală, transferul de intrare/ieșire este terminat imediat, returnîndu-se o condiție de succes sau de eroare ; nu este necesar ca task-ul emițător să aștepte terminarea intrării/ieșirii implicate în emisie.

Înainte sau după umplerea cutiei poștale, task-ul destinatar o poate examina, prin deschidere în citire, programînd o directivă Monitor **RCVD$**, **RCVX$**, **RCST$**, **VRCD$**, **VRCS$** sau **VRCX$**. În cazul în care cutia poștală este vidă, se returnează un cod de eroare. În cazul existenței unor mesaje în cutia poștală, se returnează prima intrare din cutie într-un buffer specificat de către utilizator.

În mod similar, fiecărui task din sistem i se alocă și o cutie poștală în care celelalte task-uri pot depune referințe la o regiune. Referințele sînt emise către cutia poștală prin programarea unei directive Monitor **SREF$**. În acel moment se și creează o intrare în cutia poștală a task-ului destinatar. Sistemul introduce această intrare în coada de recepție referințe a task-ului destinatar (ordonată FIFO și descrisă de locația **T.RRFL** din **TCB**), indiferent de starea task-ului (activ sau inactiv).

Examinarea cutiei poștale se poate face de către destinatar prin programarea directivei Monitor **RREF$** sau **RRST$**. În cazul în care cutia poștală este vidă, se returnează un cod de eroare (**RREF$**) sau se forțează stoparea task-ului (**RRST$**). În cazul existenței unor referințe în cutia poștală, se returnează prima intrare din cutie într-un buffer specificat de utilizator. Referința la regiune cuprinde suficiente informații (identificator regiune, deplasare în regiune, drepturi de acces, etc.) pentru a permite task-ului destinatar să-și mapeze o fereastră de adrese în regiunea specificată.

Pentru a preveni umplerea excesivă a unei cutii poștale cu mesaje (respectiv referințe), sau pentru obținerea imediată a mesajului/referinței după plasarea acestuia în cutia poștală, se pot specifica tratări **AST** pentru recepția unui mesaj sau a unei referințe la o regiune. În momentul transmiterii unui mesaj/referință către o cutie poștală, după introducerea acestuia într-una din cozile asociate, se generează o întrerupere software asincronă (**AST**) corespunzătoare.

Dacă task-ul destinatar validase deja o tratare AST la recepționarea unui mesaj (**SRDA$**) sau referință la o regiune (**SRRA$**), mesajul/referința emis(ă) este recepționat(ă) imediat, fiind extras(ă) din cutia poștală. În cazul în care nu s-a specificat o astfel de tratare, mesajul/referința este introdus(ă) în coada cutiei poștale asociate task-ului destinatar.

Cutiile poștale reprezintă de altfel în **MIX** și mecanismul de bază pentru tratarea intrărilor nesolicitate de la terminale. Utilizarea cutiilor poștale elimină în acest caz necesitatea unor citiri repetate de la terminale pentru acceptarea intrărilor nesolicitate.

Cutiile poștale implementate în sistemul de operare **MIX** admit mai mulți emițători și un singur receptor. Nu există nici o limită impusă în privința numărului de mesaje/referințe recepționate/emise de către un task.

Dimensiunea unei intrări în cutia poștală de recepție mesaje este de minimum 26 octeți (13 cuvinte) (emisie/recepție de lungime fixă) și de maximum 255 octeți (emisie-recepție de lungime variabilă).

Alocarea mesajelor (de lungime fixă sau variabilă) se face la sistemele **MIX/MIX-PLUS**, din zona secundară de memorie sistem cu alocare dinamică (**SECPOL**).

Dimensiunea unei intrări în cutia poștală de recepție referințe este de 16 octeti (8 cuvinte), iar alocarea intrărilor se face numai din zona primară de memorie sistem cu alocare dinamică.

Sînt disponibile următoarele *servicii sistem relative la cutiile poștale*:

a) mesaje de lungime fixă :
- recepție mesaj (**RCVD$**) ;
- recepție mesaj sau stopare (**RCST$**) ;
- recepție condiționată mesaj (**RCVX$**) ;
- emisie mesaj (**SDAT$**) ;
- emisie mesaj, cerere de lansare în execuție task și conectare (**SDRC$**).

b) mesaje de lungime variabilă :
- recepție mesaj (**VRCD$**) ;
- recepție mesaj sau stopare (**VRCS$**) ;
- recepție mesaj sau terminare execuție (**VRCX$**) ;
- emisie mesaj (**VSDA$**) ;
- emisie mesaj, cerere de lansare în execuție task și transmitere relații de paternitate (**SDRP$**) ;
- emisie mesaj, cerere de lansare în execuție task și conectare (**VSRC$**).

c) referințe la regiuni :
- emisie referință (**SREF$**) ;
- recepție referință (**RREF$**) ;
- recepție referință sau stopare (**RRST$**).

d) tratări **AST** :
- validare tratare întreruperi **AST** recepție mesaje (**SRDA$**) ;
- validare tratare întreruperi **AST** recepție referințe (**SRRA$**).

În subcapitolul 28.3 se prezintă diverse exemple de utilizare și programare folosind cutiile poștale.

4.3. Zone partajate

O zonă partajată reprezintă un bloc de informaţii (date) sau instrucţiuni, rezident în memorie, utilizat în comun de mai multe task-uri. Aceste zone partajate sînt extrem de utile întrucît :

— constituie un mijloc de comunicare între două sau mai multe task-uri ;

— constituie calea prin care un singur exemplar dintr-o bază de date comună sau un număr de subrutine comune sînt utilizate, concurent, de către mai multe taskuri din sistem.

În sistemul de operare **MIX** se disting mai multe tipuri de zone partajate :

— blocuri de comun globale, rezidente ;

— blocuri de comun globale, dinamice ;

— biblioteci de rutine partajate, rezidente ;

— zone reentrante ale task-urilor multiutilizator **(MIX-PLUS)**.

Termenul de „rezident“ precizează că respectiva zonă partajată a fost construită şi instalată în sistem în mod separat faţă de task-ul (urile) care se leagă de aceasta. Spre deosebire de zonele partajate rezidente, blocurile de comun globale dinamice, se creează şi se elimină dinamic la iniţiativa task-urilor ce le partajează.

Se recomandă utilizarea intensivă a acestor tipuri de zone partajate, pentru implementarea unei aplicaţii utilizator, datorită economiilor mari de memorie realizate prin aplicarea acestei soluţii. Dealtfel, majoritatea componentelor de dezvoltare de programe ale sistemului **MIX** utilizează biblioteci de rutine partajate.

În sistemele cu relocare, în care spaţiul de adresare virtual este alocat în unităţi de 4 kcuvinte, pentru economisirea spaţiului de adresare e posibilă o construire a zonei partajate rezidente cu o structură de segmente rezidente în memorie. În acest caz, task-ul nu mai este conectat la întreaga zonă partajată, ci numai la o parte din ea la un moment dat. Un exemplu important, în acest sens, îl reprezintă bibliotecile partajate de module de acces **FCS (FCSANS)** şi **RMS (RMSRES)**, construite astfel încît nu ocupă mai mult de 1 APR (4 kc) în spaţiul de adresare virtual al task-ului ce le referă.

Unei zone partajate i se asociază un fişier imagine task **(TSK)** şi un fişier cu definiţii simbolice globale **(STB)**.

În momentul în care un task se conectează la o zonă partajată, Editorul de legături **MIX** utilizează fişierul de definiţii simbolice globale ale acestei zone pentru a lega task-ul la zonele de memorie şi punctele de intrare necesare, referite din cadrul task-ului.

În sistemul de operare **MIX** încărcarea (şi respectiv fixarea) unei zone partajate, rezidente, se face la instalare, prin comanda operator **INStall.** Eliminarea acesteia este posibilă, prin comanda operator **REMove.../REG**, numai după eliminarea tuturor task-urilor ce accesează zona partajată respectivă.

În sistemul de operare **MIX-PLUS**, zonele partajate rezidente sînt manipulate dinamic. Ele se încarcă automat în memorie, la încărcarea task-urilor

ce le referă, şi pot fi evacuate implicit (în momentul în care au fost evacuate task-urile ce le referă) sau prin specificare explicită (utilizînd directiva Monitor **CPCR$**).

În sistemele de operare **MIX/MIX-PLUS**, la instalarea (statică sau dinamică) a zonelor partajate, se colectează parametrii de descriere a acestora într-o listă a blocurilor de comun (**CBD**). Vizualizarea informaţiilor din listă este posibilă cu ajutorul comenzii **MCL CBD**.

4.3.1. Blocuri de comun relocabile şi absolute

În funcţie de modul în care se încarcă în memorie, zonele partajate sînt de două feluri :

— independente de poziţie (**PIC**) (relocabile) ;
— absolute (dependente de poziţie).

Zonele partajate independente de poziţie pot fi plasate oriunde în spaţiul de adresare virtual ataşat task-ului, în cazul în care sistemul utilizează unitatea de relocare şi protecţie a memoriei.

Zonele partajate absolute trebuie să fie fixate în spaţiul de adresare virtual, întrucît la construirea acestora se precizează gama virtuală de adrese pentru care se face partajarea.

Declararea unei zone partajate ca independentă de poziţie este utilă în următoarele cazuri :

— zona partajată conţine instrucţiuni ce se vor executa corect indiferent de localizarea acesteia în spaţiul de adrese virtual al task-ului ce o referă (cod **PIC**) ;
— zona partajată conţine informaţii ce nu sînt dependente de adresare ;
— zona partajată conţine date ce vor fi referite dintr-un program scris în **FORTRAN** sau alt limbaj de nivel înalt (aceste date trebuiesc definite în comunuri cu nume).

Declararea unei zone partajate ca absolută este utilă în următoarele cazuri :

— zona partajată conţine cod (date sau instrucţiuni) ce trebuie să se găsească la o locaţie specifică în spaţiul de adresare virtual al task-ului ce o referă ;
— zona partajată conţine informaţii dependente de adresare ;
— zona partajată conţine secţiuni de program avînd nume identice cu secţiunile de program ale task-ului ce o referă.

Zonele partajate absolute sînt utilizate în cazul referirii acestora de task-uri limitate la un număr redus de registre de acces (**PAR/PDR**) sau de către componentele sistem (în acest fel se implementează driverele sistem sau utilizator încărcabile).

Declararea unei zone partajate ca independentă de poziţie se face la editarea de legături a acesteia, prin specificarea comutatorului **/PI** la fişierul imagine ataşat zonei. Pentru declararea unei zone partajate drept absolută, se utilizează comutatorul **/-PI**.

Editorul de legături **MIX** construieşte implicit o zonă partajată ca absolută. Adresa virtuală a zonei partajate absolute este dată de opţiunea **TKB PAR**.

4.3.2. Blocuri de comun globale, rezidente

Un *bloc de comun global* reprezintă o zonă de informații (date) pentru comunicare între task-uri.

Blocurile de comun globale, rezidente, se pot clasifica astfel :

— blocuri de comun statice ;

— blocuri de comun mapate în pagina externă.

În figura 4.1 este reprezentată o imagine a memoriei în care două task-uri comunică între ele prin intermediul unui bloc de comun global, rezident. În acest exemplu, task-ul A depozitează anumite rezultate în blocul C, extrase (regăsite) de task-ul B la un moment de timp ulterior.

4.3.2.1. Blocuri de comun statice

În sistemul de operare **MIX** blocurile de comun statice, rezidente, sînt tratate ca niște imagini de task-uri, cu următoarea observație : nu este necesar să fie contigue fizic cu imaginile de task-uri cu care comunică, și nici contigue între ele, atunci cînd un task comunică simultan cu mai multe blocuri de comun.

În sistemele cu relocare, fiecărui task i se atribuie un număr de registre de relocare și protecție a memoriei **(PAR/PDR)**, al căror conținut este unic per task. Aceasta înseamnă că memoria fizică a unui task nu este vizibilă de către un alt task din sistem. Nici un task nu poate citi sau scrie din/în zona de memorie rezervată altui task ; această zonă există numai pentru task-ul rezident în cadrul ei.

Blocurile de comun statice, rezidente, în schimb, sînt accesibile altor task-uri întrucît aceste zone pot fi relocate (proiectate) de către sistem în spațiile virtuale ale diferitelor task-uri.

În funcție de dimensiunea unui task utilizator (ce furnizează numărul de registre **PAR/PDR** alocate) acesta se poate lega, practic, la un număr de 1 pînă la 7 blocuri de comun. Blocurile de comun relocabile pot fi accesate, de către task-urile cooperante, cu registre de relocare și protecție a memoriei **(PAR/PDR)** diferite.

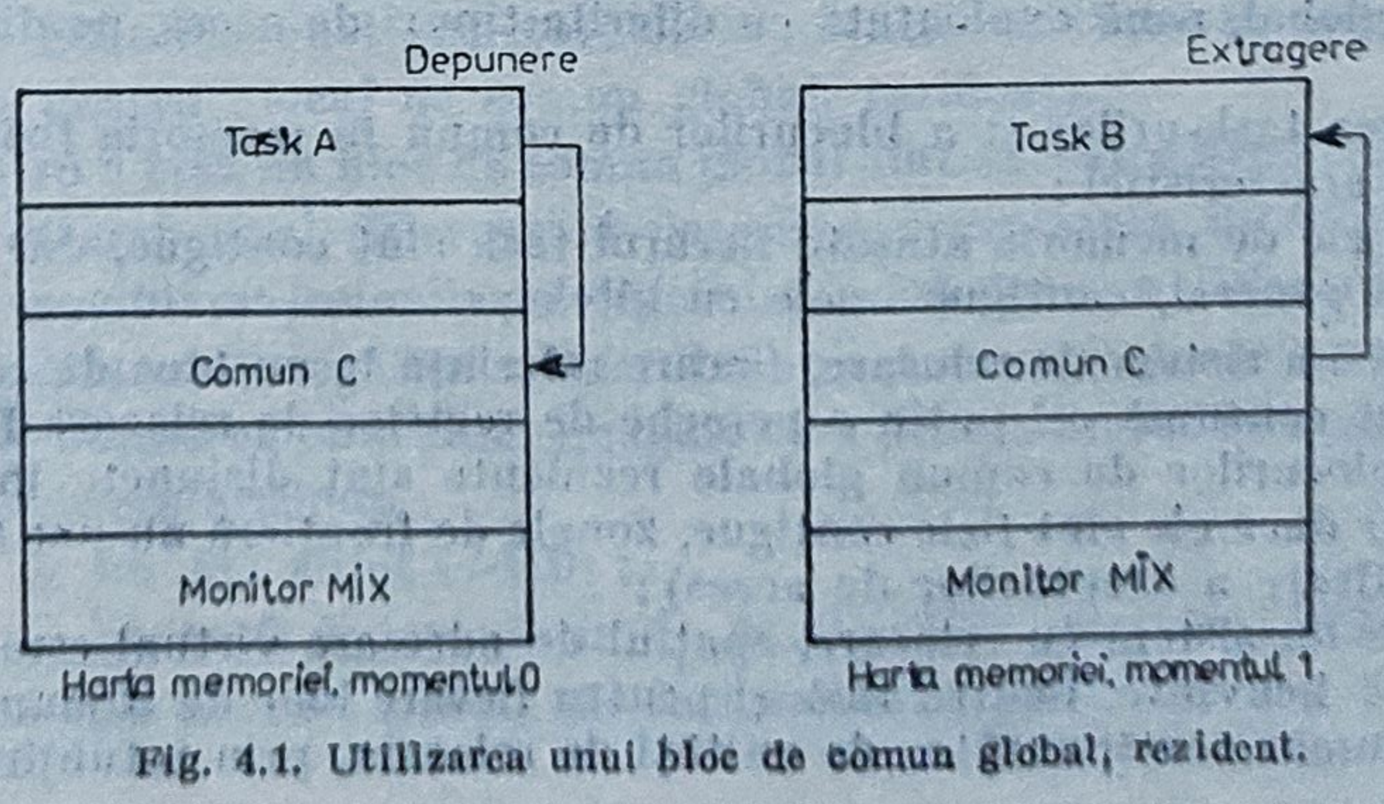


Fig. 4.1. Utilizarea unui bloc de comun global, rezident.

În sistemele fără relocare, spațiul de adresare fizic neputînd depăși 32 kcuvinte, întreaga memorie fizică este accesibilă tuturor task-urilor din sistem. În acest caz, task-urile se pot lega, teoretic, la orice număr de blocuri de comun. Pentru păstrarea integrității sistemului, se execută de către sistem o serie de validări suplimentare în momentul creerii imaginilor de task.

În general se poate considera un bloc de comun static, rezident, un fișier cu nume, avînd o anumită lungime și specificație de acces, construit sub formă de task la Editarea de legături și alocat în mod fizic, static, la reconfigurarea sistemului, sau dinamic, de către operator, înaintea primei sale utilizări.

Blocurile de comun pot fi evacuate, în sistemul de operare **MIX-PLUS**, explicit (prin programarea directivei Monitor **CPCR$**) sau implicit (la evacuarea tuturor task-urilor ce le referă).

Pentru blocurile de comun statice, cu acces numai în citire **(RO)**, evacuarea semnifică numai eliminarea acestora din memorie.

Evacuarea blocurilor (regiunilor) de comun statice sau dinamice, cu acces în citire/scriere **(RW)**, se face în fișierul sistem de evacuare cu alocare dinamică. Un caz particular, îl reprezintă acele blocuri de comun statice, cu acces în citire/scriere **(RW)**, instalate cu comanda **VMR/MCL INStall/WB = YES** sau **DCL INSTALL/WRITE-BACK**. La evacuare, acestea sînt copiate în imaginea lor de pe disc și nu în fișierul de evacuare cu alocare dinamică. Aceasta face ca orice actualizare în copia rezidentă în memorie a unei regiuni de comun să devină permanentă. Prioritatea de evacuare este identică cu cea a task-ului mapat la regiunea de comun.

Adresarea unui bloc de comun se face serial, par task, cu următoarele drepturi de acces :

— numai în citire **(RO)** ;

— în citire/scriere **(RW)**,

specificate la editarea de legături a task-ului ce urmează a se conecta la blocul de comun (opțiunile **COMMON** și **RESCOM**).

Între mai multe task-uri, partajarea unui bloc de comun poate fi făcută parțial sau total, cu diferite drepturi de acces.

Figura 4.2 reprezintă un bloc de comun partajat parțial de către trei task-uri. Task-ul **A** partajează blocul de comun pe porțiunea C_A numai în citire, task-ul B pe porțiunea C_B în citire/scriere, iar task-ul **C** pe porțiunea C_C numai în citire. O astfel de situație este extrem de frecventă în cazul programelor scrise în **FORTRAN**, ce au zona de **COMMON** plasată într-un bloc de comun global, zonă exploatată cu diferite tipuri de acces, pe diferite porțiuni.

Alocarea task-urilor și a blocurilor de comun în memoria fizică are următoarele caracteristici :

— zonele de memorie atașate fiecărui task sînt contigue, dar task-urile nu sînt, în general, contigue unele cu altele ;

— într-un sistem de relocare, fiecare referință la un bloc de comun global rezident consumă cel puțin o pereche de registre de relocare **PAR/PDR**. Alocările blocurilor de comun globale rezidente sînt disjuncte în memoria fizică (chiar dacă ele sînt fizic contigue, zonele de frontieră nu pot fi depășite fără o încălcare a drepturilor de acces) ;

— într-un sistem cu relocare, spațiul de adresare virtual este alocat în unități de 4 kcuvinte. Pentru task și pentru fiecare bloc de comun referit se alocă un număr corespunzător de unități de relocare, prin rotunjirea dimen-

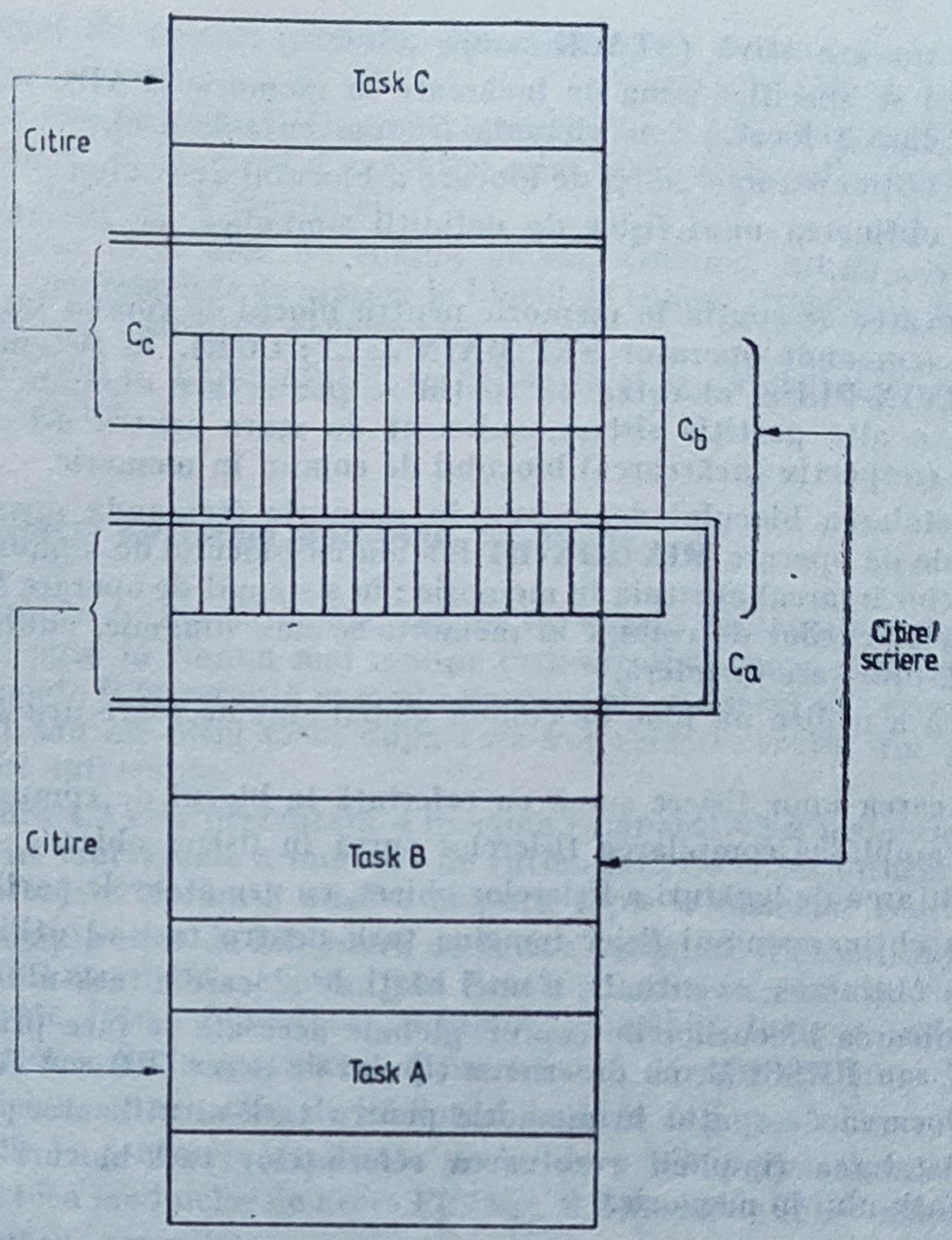


Fig. 4.2. Partajarea unui bloc de comun global, rezident, cu mai multe tipuri de acces

siunii fiecăruia la multiplul superior de 4 kcuvinte. În cazul în care spațiul de adresare virtual depășește 32 kcuvinte, e necesară o reproiectare a aplicației prin definirea unei structuri de segmente rezidente în memorie (pentru task sau pentru blocul de comun global, rezident).

Pentru a crea un bloc de comun global sînt necesare următoarele acțiuni :

— crearea unui fișier sursă descriind organizarea blocului de comun ;
— asamblarea/compilarea fișierului sursă într-un fișier obiect ;
— editarea de legături a fișierului obiect, cu următoarele particularități :

a) obținerea unui fișier imagine task pentru blocul de comun, cu următoarele atribute :

— nu are antet (**/-HD**) ;
— poate fi independent de poziție (**/PI**) sau nu (**/-PI**) ;
— cu relocare (**/MM**) sau fără relocare (**/-MM**) ;

— nu are stivă (**STACK = 0**);
— i se specifică zona de încărcare în memorie (**PAR**), reală (sistem fără relocare) sau virtuală (sistem cu relocare).

b) obținerea unei hărți de alocare a blocului de comun ;
c) obținerea unui fișier de definiții simbolice, ce descrie blocul de comun.

— alocarea de spațiu în memorie pentru blocul de comun într-o partiție de comun (comanda operator **SET/MAIN = ... : COM**). În sistemele de operare **MIX/MIX-PLUS**, alocarea de spațiu se poate face dinamic, în partiția **GEN** (sau o altă partiție sistem, suficient de mare pentru a-l conține), la instalarea (respectiv încărcarea) blocului de comun în memorie.

— instalarea blocului de comun în memorie (comanda operator **INS**). În sistemele de operare **MIX/MIX-RT** instalarea blocului de comun semnifică și fixarea (încărcarea) acestuia în memorie ; în sistemul de operare **MIX-PLUS**, încărcarea blocurilor de comun în memorie se face dinamic, odată cu încărcarea task-ului care le referă.

Pentru a utiliza un bloc de comun global sînt necesare următoarele acțiuni :

— crearea unor fișiere sursă cu referință la blocul de comun ;
— asamblarea/compilarea fișierelor sursă în fișiere obiect ;
— editarea de legături a fișierelor obiect, cu următoarele particularități :

a) obținerea unui fișier imagine task pentru task-ul utilizator ;
b) obținerea, eventuală, a unei hărți de alocare a task-ului utilizator.

Specificarea blocurilor de comun globale accesate se face prin opțiunea **COMMON** sau **RESCOM**, cu descrierea tipului de acces (**RO** sau **RW**).

— alocarea de spațiu în memorie pentru task-ul utilizator ;
— instalarea (implicit rezolvarea referințelor task-blocuri de comun referite) task-ului în memorie.

4.3.2.2. Blocuri de comun mapate în pagina externă

Față de schema prezentată anterior, trebuie făcută o mențiune specială relativă la crearea și referirea blocurilor de comun rezidente în pagina externă (ultimele 4 kcuvinte de memorie, ce reprezintă registrele de stare și control ale dispozitivelor periferice).

În acest caz, alocarea de spațiu în memorie se face fictiv, într-o partiție de comun plasată în pagina externă (comandă operator **SET/MAIN =... : DEV**), iar instalarea blocului de comun în memorie nu semnifică fixarea.

În sistemul de operare **MIX-PLUS**, declararea partiției de comun se face prin comanda operator generală **SET/PAR = ... : SYS**, identificarea paginii externe făcîndu-se prin adresele sale virtuale. Orice utilizare specifică, partajată, a partiției de comun plasate în pagina externă, duce la crearea unei subpartiții atașate partiției principale (de tip **DEVICE**). În acest caz, nu este necesară instalarea blocului de comun în memorie. Spre deosebire de blocurile de comun statice, cele mapate în pagina externă nu se evacuează.

4.3.3. Blocuri de comun globale, dinamice

Un bloc de comun global, dinamic, reprezintă o zonă de memorie (**regiune**) creată dinamic, în momentul execuției unui task, utilizînd directive Monitor speciale de gestiune a memoriei.

Partajarea unui bloc de comun global, dinamic, are loc numai după acceptarea unei **referințe de atașare** la blocul de comun (transmisă și, respectiv, recepționată de task-urile cooperante).

În capitolul 5, paragraful 7, se definește și se explicitează conceptul de regiune dinamică.

4.3.4. Biblioteci de rutine partajate, rezidente

O bibliotecă de rutine partajate, rezidente, constă dintr-un set de subrutine reentrante, puse în comun mai multor task-uri din sistem. Execuția acestor subrutine poate fi întreruptă în mod asincron; ele vor servi alte cereri ale task-ului curent sau ale altui task, după care vor relua execuția din punctul în care au fost întrerupte.

În figura 4.3 este prezentată o imagine comparativă a memoriei în cazul unei utilizări individuale a unui set de rutine față de cazul utilizării unei biblioteci partajate (formată dintr-o singură copie a rutinelor comune).

Un exemplu tipic de bibliotecă de rutine partajate îl constituie modulele de acces **FCS** sau **RMS**. Acestea pot fi atît atașate fiecărui task în sistem, cît și grupate într-o bibliotecă partajată, accesibilă tuturor task-urilor ce utilizează sistemul de gestiune a fișierelor sau înregistrărilor.

Dimensiunea unei astfel de biblioteci depinde de numărul de rutine selectate, stabilit la generare (dacă este vorba de o bibliogecă sistem — ca, de pildă biblioteca modulelor de acces **FCS** sau **RMS**) sau de către utilizator (înainte utilizării acesteia).

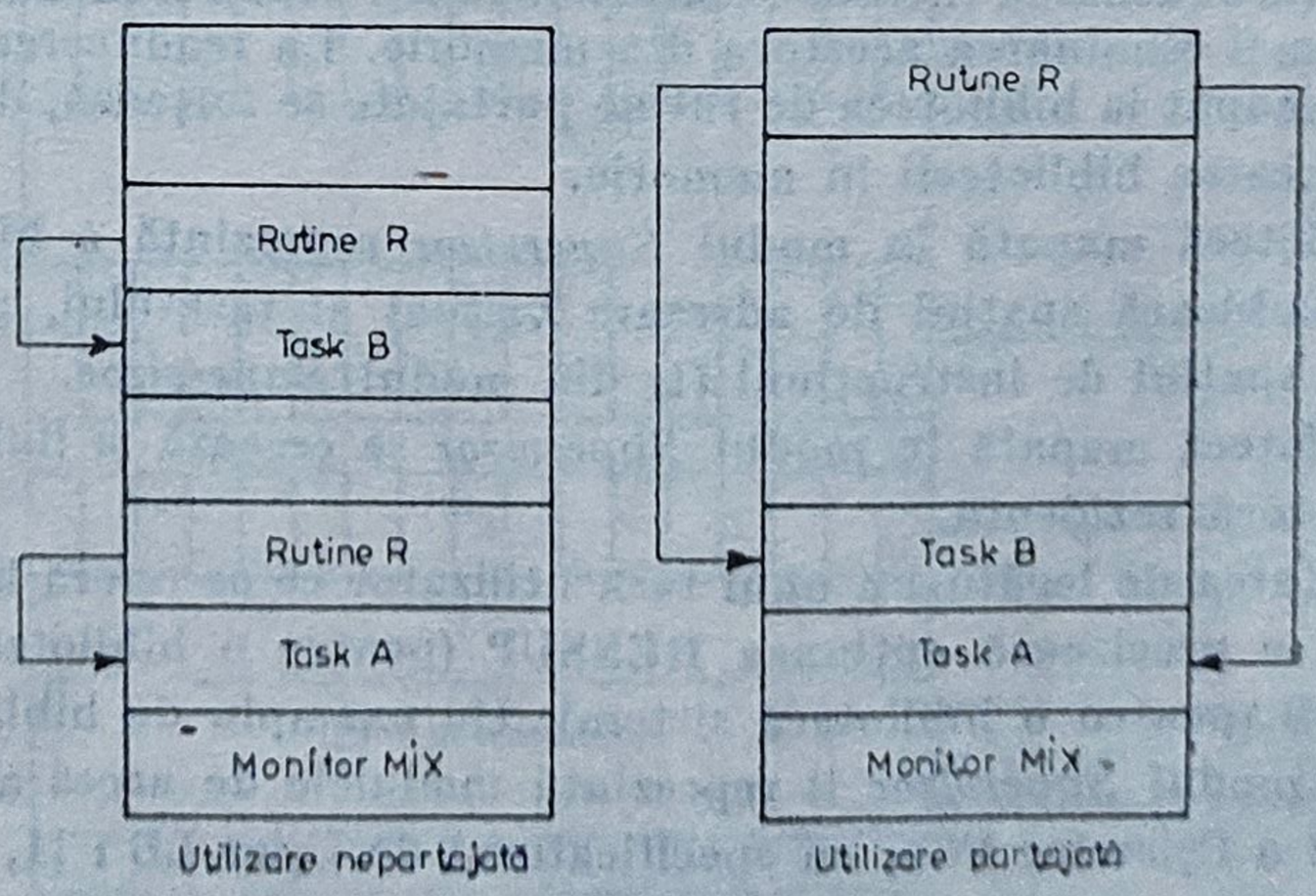


Fig. 4.3. Utilizarea unei biblioteci de rutine partajate

O bibliotecă de rutine partajate o dată constituită, devine o funcție rezidentă permanentă a sistemului, disponibilă tuturor task-urilor ce se referă la una din funcțiile conținute în bibliotecă. Într-un sistem de operare **MIX** pot fi constituire oricîte biblioteci de rutine partajate, cu nume, fiecare compusă din diferite module funcționale, cu condiția ca toate să fie rezidente în momentul referirii acestora.

Ca și blocurile de comun, bibliotecile de rutine partajate, rezidente, sînt tratare ca niște imagini de task-uri, păstrîndu-se observațiile făcute la blocurile de comun, rezidente. Spre deosebire de blocurile de comun, adresarea unei astfel de biblioteci se face concurent (datorită reentranței rutinelor componente), numai în citire **(RO)**.

Alocarea task-urilor și a bibliotecilor de rutine partajate, rezidente în memoria fizică, are următoarele caracteristici :

— într-un sistem cu relocare, fiecare referință la o bibliotecă de rutine partajate consumă cel puțin o pereche de registre de relocare **PAR/PDR**. Dacă biblioteca depășește 4 kcuvinte, se alocă noi registre de relocare, unul pentru fiecare increment de 4 kcuvinte ;

— dacă biblioteca referită este independentă de poziție **(PIC)**, e necesară numai o pereche de registre **PAR/PDR** pentru a o accesa ;

— dacă biblioteca referită este absolută, alocarea fiecărei perechi **PAR/PDR** pentru un task trebuie să fie identică cu alocarea efectuată pentru alte task-uri.

Pentru crearea unei biblioteci de rutine partajate, rezidente, sînt necesare aceleași acțiuni ca pentru crearea unui bloc de comun, rezident. Pentru utilizarea bibliotecii partajate de către un task sînt necesare acțiuni asemănătoare ca pentru blocul de comun, cu următoarele particularități :

— specificarea bibliotecii accesate se face, la editarea de legături, cu opțiunea **LIBR** sau **RESLIB** ;

— tipul de acces la bibliotecă este întotdeauna numai în citire **(RO)**.

Bibliotecile de rutine partajate, rezidente, pot fi evacuate, în sistemul de operare **MIX-PLUS**, explicit (prin programarea directivei Monitor **CPCR$**) sau implicit(la evacuarea tuturor task-urilor ce le referă). Evacuarea semnifică, de fapt, numai eliminarea acestora din memorie. La readucerea în memorie a task-ului mapat la biblioteca de rutine partajate se forțează, de către Monitor, și încărcarea bibliotecii în memorie.

O bibliotecă mapată în modul *Supervizor* reprezintă o bibliotecă rezidentă, ce dublează spațiul de adresare virtual al task-ului, prin maparea acesteia în spațiul de instrucțiuni **(I)** din modul *Supervizor*.

O bibliotecă mapată în modul *Supervizor* se creează și link-editează ca orice bibliotecă rezidentă.

La editarea de legături a unui task utilizator ce se referă la o asemenea bibliotecă, se precizează opțiunea **RESSUP** (pentru o bibliotecă utilizator) sau **SUPLIB** (pentru o bibliotecă sistem). Un exemplu de bibliotecă sistem mapată în modul *Supervizor* îl reprezintă modulele de acces ale sistemului de gestiune a fișierelor **FCS**, cu specificatorul de fișier **LB : [1, 1] FCSFSL. TSK.**

4.3.5. Zone reentrante ale task-urilor multiutilizator

În sistemul de operare **MIX-PLUS**, un task multiutilizator constă dintr-o *zonă pură* (reentrantă) și o *zonă impură* (informații și date nereentrante).

Zona reentrantă este acea zonă nemodificată în timpul execuției task-ului și care poate fi partajată între mai multe versiuni ale aceluiași task.

Zona impură, nereentrantă, se schimbă la fiecare execuție a task-ului; la un moment dat, în memorie există cîte o copie a zonei impure pentru fiecare utilizator simultan al task-ului.

În figura 4.4 este reprezentată alocarea memoriei în cazul existenței unui task multiutilizator. Zonele impure I_A, I_B și I_C interacționează simultan cu zona reentrantă TR a task-ului multiutilizator.

În sistemul de operare **MIX-PLUS**, o bună parte din componentele sistem (task-uri sistem, compilatoare, programe utilitare) sînt implementate sub această formă, separînd la asamblare/editare de legături zonele de cod (reentrante) de cele de date (nereentrante). Mecanismul de utilizare partajată a zonei de cod pure este și mai performant în condițiile existenței reale a celor două spații fizice de implementare (date și instrucțiuni). Pentru minicalculatoarele românești **I-102F**, **I-102/4M**, **I-106** și **I-1016**, avînd implementată o nouă unitate de relocare și protecție a memoriei, care să recunoască spații de date și instrucțiuni fizic diferite, este posibilă utilizarea reală a acestei facilități.

Crearea de task-uri multiutilizator duce la reducerea substanțială a necesarului de memorie pentru o aplicație dată și la obținerea unor performanțe de exploatare mai bune pentru sistemul de operare.

La evacuarea din memorie a unei copii de task multiutilizator, zona sa pură nu se evacuează. În cazul evacuării tuturor copiilor unui task multiutilizator, zona sa pură se evacuează prin eliminarea din memorie. Aducerea în memorie a oricărei copii forțează și reîncărcarea zonei pure, de pe disc, din imaginea prototip a task-ului multiutilizator.

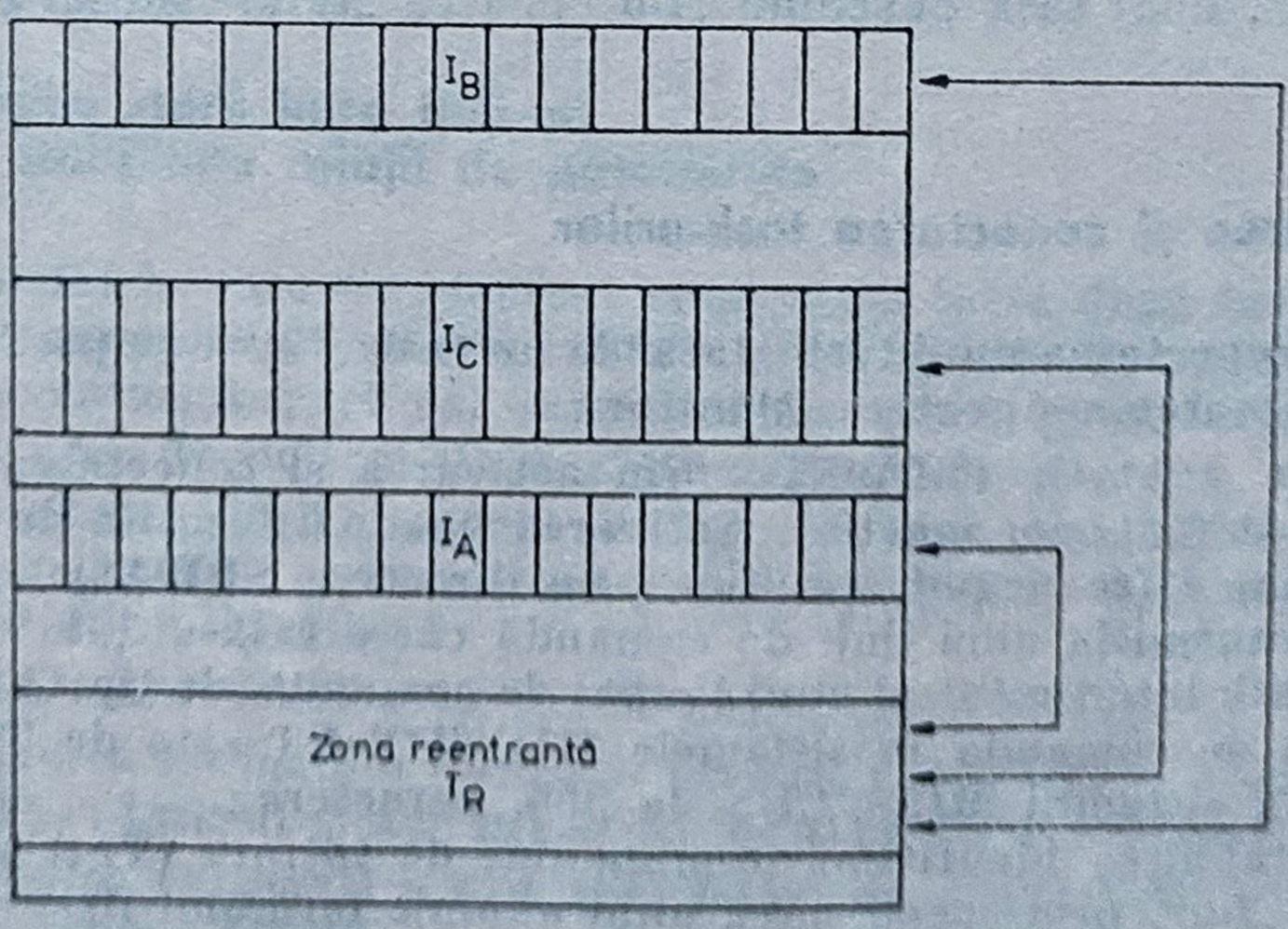


Fig. 4.4. Alocarea memoriei pentru task-urile multi-utilizator

4.4. Relații de paternitate între task-uri

În cadrul mecanismului general de multitasking, familia de sisteme de operare **MIX** suportă și facilitatea de a crea și controla relații complexe între task-uri (relații de paternitate), ce se stabilesc între task-uri „tată" și task-uri „fiu".

Un task „tată" este acel task care creează relațiile de paternitate prin lansarea în execuție și conectarea la un alt task, numit task „fiu" (sau subtask). La terminarea execuției task-ului „fiu", task-ul „tată" este atenționat și poate primi informații de stare din partea task-ului „fiu". Starea returnată în acest caz poate indica terminarea cu succes a execuției task-ului „fiu" sau identifică apariția unor condiții de eroare specifice.

Sincronizarea execuției între task-urile „fii" și „tată" are loc ca urmare a inițierii execuției (activării) task-urilor „fii" de către task-ul „tată".

Comunicarea între task-urile „fii" și „tată" se face prin conectarea/deconectarea unui task „tată" la/de la un task „fiu" și prin transmiterea de stare efectuată de task-urile „fii" către task-ul „tată".

Un task „tată" poate activa și se poate conecta la mai multe task-uri „fii", de asemenea, un task „tată" se poate conecta multiplu la același task „fiu". În schimb, un task „fiu" poate fi conectat la mai multe task-uri „tată".

O operație importantă a relațiilor de paternitate între task-uri o reprezintă, în sistemul de operare **MIX-PLUS**, subsistemul de prelucrări batch ; relațiile și parametrii asociați task-urilor pot fi stabilite on-line, iar controlul execuției uneia sau mai multor lucrări batch are loc off-line.

Comunicarea și sincronizarea între task-uri care au stabilite relații de paternitate se face prin utilizarea unor directive Monitor specifice (**SPWN$, CNCT$, SDRC$, RPOI$** sau **SDRP$**). Aceste directive se pot împărți în două clase :

— de activare și creare a conexiunii între task-urile „tată" și „fii" (**spawning** — creare subtask) ;

— de transfer, către un alt task, a conexiunii stabilite între un task „tată" și „fiu", noul task devenind „fiu" pentru „tatăl" inițial (**chaining** — înlănțuire).

4.4.1. Activarea și conectarea task-urilor

Activarea și conectarea unui task „tată" la un task „fiu" este posibilă prin utilizarea următoarelor directive Monitor :

— creare subtask (**SPWN$**), prin activarea și conectarea la un task „fiu" specificat (anterior inactiv). Activarea poate fi însoțită de următoarele acțiuni opționale (ce se pot specifica prin directiva **SPWN$**):

— transmisia unei linii de comandă către task-ul „fiu" (ce poate fi un task interpretor al unui limbaj de comandă, de tip **CLI**) ; lungimea liniei de comandă în sistemele **MIX/MIX-RT** este de 79. caractere, iar în sistemul **MIX-PLUS** de 255. caractere ;

— stabilirea identității terminalului de intrare (**TI :**) asociat task-ului „fiu", prin specificarea unui anumit terminal fizic sau, în cazul sistemului **MIX-PLUS**, a unui terminal virtual creat anterior.

— conectare la subtask (**CNCT$**), în cazul în care acesta este deja activ. Conectarea permite sincronizarea task-ului apelant („tată") cu terminarea (normală sau anormală) sau emiterea de stare a task-ului „fiu" ;

— emisie de date către un subtask, activarea acestuia (dacă nu era deja activ) și conectarea la subtask (**SDRC$**).

La activarea sau conectarea task-ului „fiu", Monitorul atașează acestuia un bloc de control al conexiunii (**OCB**), ce conține toate informațiile necesare utilizării conexiunii și transmiterilor de stare între task-uri. La fiecare nouă conectare, se creează un nou bloc **OCB** pentru task-ul „fiu" ; coada blocurilor **OCB** (ordonată FIFO) asociată task-ului „fiu" este descrisă de locația **T.OCBH** din **TCB**.

După conectare, modalitatea specifică de stabilire a acesteia nu mai poate fi identificată de către Monitor. Sincronizarea execuției între task-urile „tată" și „fii" are loc, de obicei, prin utilizarea directivelor **STSE$**, **STLO$**, **STOP$S** și **USTP$S**.

4.4.2. Transferarea informațiilor de conectare între task-uri

După stabilirea conexiunii între un task „tată" și „fiu", aceasta poate fi transferată unui nou task (ce devine „fiu" pentru task-ul „tată"), prin utilizarea următoarelor directive Monitor :

— lansarea în execuție a unui task specificat și înlănțuire, prin transmitere către task a uneia sau mai multor conexiuni (**RPOI$**), stabilite anterior între task-ul curent și tatăl acestuia. Prin transmiterea conexiunilor, task-ul specificat devine un task „fiu" pentru „tatăl" ce a inițiat aceste conexiuni. Opțional, înlănțuirea poate fi însoțită de transmisia unei linii de comandă către noul task „fiu" (ce poate fi un task interpretor al unui limbaj de comandă) ; lungimea liniei de comandă în sistemele **MIX/MIX-RT** este de 79. caractere, iar în sistemul **MIX-PLUS** de 255. caractere.

— emisie de date către un task specificat, activarea acestuia (dacă nu era deja activ) și înlănțuire (**SDRP$**).

4.4.3. Transmiterea stării între task-uri care au stabilitate relații de paternitate

Indiferent de modul în care s-a efectuat conectarea între două task-uri, un task „fiu" poate transmite informații de stare către un task „tată", poate seta un indicator de eveniment specificat de un task „tată" sau poate forța apariția unei întreruperi **AST**, de emitere stare, către un task „tată".

După transmiterea stării efectuată de către un task „fiu", către unul sau mai multe task-uri „tată", acestea (task-urile „tată") sînt deconectate de la task-ul „fiu" de către Monitor.

Transmiterea stării poate avea loc în următoarele situații :

— la terminarea normală a execuției task-ului „fiu" (prin programarea directivei **EXIT$S**) ; task-ul „tată" primește codul de stare **EX$SUC** ;

— la terminarea anormală a execuției task-ului „fiu" ; task-ul tată primește codul de stare **EX$SEC** ;

— la terminarea (normală sau anormală) a task-ului „fiu“ cu o stare specifică asociată (prin programarea directivei **EXST$**); toate task-urile „tată“ la care acesta este conectat primesc codul de stare precizat de directiva **EXST$**;

— la emiterea de stare a unui task „fiu“ către un task „tată“ cunoscut (specificat) sau necunoscut (prin programarea directivei **EMST$**):

— pentru un task „tată“ cunoscut, informațiile de conectare asociate acestuia se culeg din coada blocurilor de control ale conexiunilor (**OCB**), ordonată FIFO, asociată task-ului „fiu“ (ce emite starea). În cazul în care un task „tată“ este multiplu conectat la un task „fiu“, se utilizează conexiunea cea mai veche (prima din coada **OCB**). Blocul **OCB** selectat este folosit pentru transmiterea de stare către task-ul „tată“;

— pentru un task „tată“ neprecizat (necunoscut), emiterea de stare se face către toate task-urile „tată“ conectate la task-ul „fiu“.

Indiferent de identitatea task-ului „tată“, blocul **OCB** asociat acestuia poate conține un indicator de eveniment și adresa unei rutine de tratare **AST** emisie stare. Dacă acestea sînt specificate, la emiterea de stare, task-ul fiu setează indicatorul de eveniment și forțează apariția unei întreruperi **AST** asociate.

În cazul în care un task „tată“ și-a terminat execuția înainte de emiterea de stare de către un task „fiu“, acțiunile anterioare nu mai au loc. Această procedură este utilă pentru task-urile multiplu conectate, în vederea păstrării sincronizării, chiar dacă task-urile „tată“ și-au terminat execuția în mod neașteptat.

Starea emisă de către un task „fiu“ către un task „tată“ este specifică relațiilor stadilite între aceste task-uri, putînd avea orice valoare cuprinsă pe 16 biți.

De obicei, se programează această stare cu următoarele valori: **EX$WAR** = 0 — Atenționare (execuția task-ului s-a terminat normal, dar pot apare erori ulterioare); **EX$SUC** = 1 — Succes (rezultatele sînt cele așteptate); **EX$ERR** = 2 — Eroare (rezultatele nu sînt cele așteptate); **EX$SEV** = 4 — Eroare gravă (execuția task-ului s-a terminat anormal).

4.5. Sincronizarea execuției task-urilor prin bitul de stopare

Sincronizarea prin bitul de stopare permite unuia sau mai multor task-uri să fie evacuat(e) (local sau la distanță) pe durata unei introduceri de date (buferate) de la un terminal sau în cazul blocării (autostopării) pe completarea unui eveniment (deblocarea urmînd a avea loc la setarea indicatorului de eveniment asociat sau în cazul execuției directivei Monitor **USTP$**).

Sincronizarea poate fi controlată prin setarea/ștergerea bitului de stopare din blocul **TCB** asociat task-ului.

La setarea bitului de stopare task-ul este blocat, prioritatea sa (cu care concură la alocarea resursei memorie internă) este coborîtă la zero și task-ul poate fi evacuat de către oricare alt task din sistem, indiferent de prioritate. Odată evacuat, task-ul va concura la alocarea resursei memorie numai după ștergerea bitului de stopare. În acest caz, task-ul va fi destopat, prioritatea sa normală va fi restaurată și va putea concura la alocarea memoriei pe baza acestei priorități.

Dacă un task stopat primește o întrerupere asincronă (**AST**), el va fi destopat și activat pe durata execuției rutinei de tratare **AST** pe baza priorității anterioare stării de stopare. La ieșirea din rutina de tratare **AST**, dacă nu există alte întreruperi **AST** în așteptare, task-ul este stopat din nou și prioritatea sa de execuție scade la zero. Pe durata execuției rutinei de tratare **AST** un task nu poate fi stopat ; orice directivă de stopare este ilegală în starea **AST**. Dacă în starea **AST** se emit cereri de I/E buferate, Monitorul nu stopează task-ul și nu buferează I/E specificată. În rutina de tratare **AST**, task-ul poate forța destoparea sa numai prin programarea unei directive **USTP$** sau **SETF$** (destoparea devenind efectivă numai după terminarea rutinei de tratare **AST — ASTX$S**).

Există *trei posibilități pentru un task neprivilegiat de a fi stopat* și, respectiv, *destopat* (la un moment dat, se poate aplica numai una din aceste metode) :

— Un task poate fi stopat la inițierea unei operații de intrare/ieșire buferate, pe durata introducerii de date de la un terminal sau, în cazul unui task „fiu", la emiterea unei operații de intrare/ieșire către un terminal virtual. Un astfel de task nu poate fi destopat decît la terminarea operației specifice de I/E ;

— Un task poate fi stopat pe unul sau mai mulți indicatori de evenimente prin programarea directivelor Monitor **STSE$** sau **STLO$**. În acest caz, task-ul poate fi destopat numai prin setarea indicatorului sau indicatorilor de evenimente corespunzător(i) ;

— Un task poate fi stopat prin programarea unei directive Monitor **STOP$S**, **RCST$ VRCS$**, **RRST$** sau **GCCI$**. În acest caz, task-ul poate fi destopat prin execuția unei directive Monitor **USTP$**, comenzii operator **MCL UNStop** sau comenzii operator **DCL START**.

Pentru utilizarea sincronizării prin bitul de stopare se folosesc următoarele directive Monitor :

— autostopare task (**STOP$S**) ;

— autostopare task în cazul inexistentei unor mesaje/referințe în coada de recepție asociată task-ului (**RCST$**, **VRCS$** sau **RRST$**) ;

— autostopare multiplă task pe indicatori de evenimente (**STLO$**), dacă nici unul din aceștia nu este setat ; în caz contrar, task-ul nu este stopat ;

— autostopare task pe indicator de eveniment (**STSE$**), dacă acesta nu este setat ; în caz contrar, task-ul nu este stopat ;

— destopare task, anterior stopat ca urmare a execuției unei directive **STOP$**, **RCST$** sau **VRCS$** (**USTP$**) ;

— stoparea unui task interpretor de comenzi (**CLI**) în absența unei linii de comandă asociate (**GCCI$**).

4.6. Terminale virtuale

Terminalele virtuale reprezintă, sub sistemul de operare **MIX-PLUS**, pseudo-dispozitive similare cu terminalele fizice. Spre deosebire de acestea, terminalele virtuale nu sînt interactive. Un utilizator nu poate solicita introduceri de date și nu poate primi secvențe de identificare (prompter), neexistînd o interacțiune directă între utilizator, terminal și task-ul aflat în execuție.

Crearea și eliminarea terminalelor virtuale are loc la inițiativa task-urilor „tată", cu ajutorul unor directive Monitor specifice :

— **CRGF$** — creare terminal virtual ;
— **ELGF$** — eliminare terminal virtual.

Terminalele virtuale sînt utilizate, de exemplu, în prelucrările batch sau similare cu acestea, asigurînd un suport de I/E tip terminal pentru subtask-uri ce ar fi solicitat, în mod normal, intervenții utilizator sau operator.

Subtask-urile, sau task-urile „fiu", create de sau conectate la un task „tată", ce a solicitat crearea unui terminal virtual, pot efectua operații de intrare/ieșire cu terminalul virtual în mod similar cu cele de pe terminalele fizice.

Terminalele virtuale diferă de terminalele fizice prin aceea că ele efectuează intrări/ieșiri de date cu un program (task „tată") și nu cu un dispozitiv de I/E fizic.

4.6.1. Crearea și eliminarea terminalelor virtuale

Directiva Monitor **CRVT$** creează un terminal virtual utilizat pentru comunicarea între un task „tată" și un task „fiu". La emiterea directivei, se creează structurile de date asociate terminalului virtual **(DCB, UCB, SCB)**, se introduc în listele sistem corespunzătoare și se asociază noului terminal virtual cel mai mic număr liber de unitate de terminal. Numerotarea terminalelor virtuale începe de la 1, structura de date a terminalului zero **(VT0)** fiind folosită ca prototip pentru crearea dinamică a structurilor de date asociate noilor terminale ; numărul maxim de terminale virtuale este de 255.

Numărul de terminal virtual, returnat în **$DSW**, urmează a fi ulterior folosit în operațiile de intrare/ieșire efectuate între task-urile „tată" și „fiu". Numele și numărul de terminal virtual se pot deasemenea preciza în directivele de creare și conectare la subtask-uri (**SPWN$, RPOI$, CNCT$**, etc.) În acest caz, subtask-ul va primi drept terminal de intrare (**TI :**) unitatea de terminal virtual precizată în apelul de directivă.

Eliminarea unui terminal virtual se face prin directiva Monitor **ELVT$** (ce dezalocă și structura de date asociată acestuia).

4.6.2. Interfațarea și utilizarea terminalelor virtuale

Terminalele virtuale sînt utilizate *direct* (prin directive **QIO$** cu funcții generale sau specifice) și *indirect*, prin directivele Monitor ce precizează o unitate de terminal virtual.

Efectuarea de operații de intrare/ieșire pe terminale virtuale (între două task-uri cooperante, aflate în relația „tată" „fiu") este posibilă, sub sistemul

de operare **MIX-PLUS**, prin intermediul driver-ului de terminal virtual **(VTDRV).**

O cerere de citire/scriere emisă de către un task „fiu", pe propriul terminal TI:, este transmisă de driverul de terminal virtual către task-ul „tată", prin intermediul unității de terminal virtual create. Task-ul „tată" răspunde printr-o cerere complementară de scriere/citire pe respectiva unitate de terminal virtual.

Driverul de terminal virtual nu interpretează sau modifică contorul și șirul de octeți transferați, codurile subfuncțiilor de I/E sau caracterele de format vertical. În cazul existenței unei neconcordanțe între lungimile de transfer precizate de task-urile „tată" și „fiu", se trunchiază transferul de date pe lungimea minimă precizată.

La terminarea unei operații de I/E pe un terminal virtual se forțează o întrerupere software asincronă **(AST),** dacă se precizează acest lucru la crearea terminalului virtual **(CRVT$).** Utilizarea mecanismului de întreruperi **AST** asigură serializarea efectivă a răspunsurilor task-ului „tată" la cererile task-furilor „fii".

Pentru ca task-urile implicate în procesul de comunicare să poată fi stopate și evacuate din memorie este posibilă buferarea datelor transferate în zona Monitor cu alocare dinamică secundară **(SECPOL).** Lungimea maximă de buferare este de 16 Kó.

4.7. Fișiere partajate

Fișierele partajate sînt utilizate de aplicațiile cooperante ce lucrează cu volume extrem de mari de informații.

Sincronizarea accesului între task-urile emițătoare și receptoare este specifică aplicației utilizator. Accesul la informațiile din fișierele partajate se face la viteza de acces a dispozitivelor disc ce conțin aceste informații.

5

Gestiunea memoriei și spațiii de adresare

Restricțiile de memorie au impus, în sistemul de operare **MIX**, atît găsirea unor soluții de utilizare eficientă a memoriei (proiectarea unor structuri de date și programe eficiente, mecanisme de reutilizare a memoriei, alocări statice și dinamice de memorie etc.) cît și o anumită structurare a acesteia între componentele sistem și utilizator.

Structurarea memoriei și modul de gestionare a acesteia sînt influențate în mod direct de prezența dispozitivului de relocare și protecție a memoriei în cadrul Unității centrale.

5.1. Sisteme de operare MIX cu și fără relocare

Sistemul de operare **MIX** este destinat să lucreze pe toate modelele de minicalculatoare românești. Structura de cuvînt pe 16 biți a minicalculatorului românesc limitează posibilitățile de adresare directă ale memoriei la 32kcuvinte (65 536 octeți).

Un astfel de sistem, ce nu utilizează dispozitivul de relocare și protecție a memorie, este numit *sistem nemapat (sistem fără relocare)*. Task-urile ce se execută pe sisteme nemapate au acces la toată memoria fizică existentă pînă la limita de 32 kcuvinte.

Singurul minicalculator românesc ce *nu* include un dispozitiv de relocare și protecție a memoriei este *CORAL 4001(A)*. În familia de sisteme de operare **MIX**, numai sistemul **MIX-RT** poate fi generat și ca un sistem nemapat.

Pentru utilizarea unor memorii mai mari de 32 kcuvinte toate minicalculatoarele românești sînt dotate cu un dispozitiv de relocare și protecție a memoriei.

În cazul utilizării dispozitivului de relocare și protecție a memoriei, adresa conținută într-un cuvînt de 16 biți nu mai este interpretată ca o *adresă fizică (AF)*, ci ca o *adresă virtuală (AV)*, ce conține informații pentru

construirea unei noi adrese fizice pe 18 biți (pentru minicalculatoarele I-100, I-102F, **CORAL 4011(A)**) sau 22 biți (pentru minicalculatoarele I-102F cu extensie de memorie, **I-102/4M, I-106, I-1016, CORAL 4030, CORAL 4021, CORAL 4015 şşi CESAR-16).**

Spațiul de adresare virtual este divizat în pagini de 4 kcuvinte. Fiecare pagină poate fi relocată într-un spațiu de adresare fizic separat.

Informația conținută în *adresa virtuală* este combinată cu informațiile de relocare și descriere conținute într-un set de *registre de pagină* **(APR),** pentru obținerea adresei fizice.

Fiecare *registru de pagină* **(APR)** este format din două registre de 16 biți : un *registru de adresă al paginii* **(PAR)** și un *registru de descriere al paginii* **(PDR).**

În sistemele de operare **MIX** există *două spații virtuale de adresare : Monitor (Kernel)* și *Utilizator (User)*, cărora le corespund două seturi a cîte 8 registre de pagină. Alegerea unuia sau altuia dintre spații este determinată de modul curent de lucru descris de Starea Program **(PS).**

Pe minicalculatoarele românești **I-102F** cu extensie de memorie, **I-102/4M, I-106, I-1016** se pot separa efectiv zonele de date **(D)** și instrucțiuni **(I),** existînd în mod corespunzător 4 seturi a cîte 8 registre de pagină pentru cele două spații virtuale de adresare. În plus, minicalculatoarele românești **I-106** și **I-1016,** dispun de un mod de lucru suplimentar : *Supervizor,* pentru care există în mod corespunzător alte 2 seturi a cîte 8 registre de pagină. Aceste facilități hardware nu pot fi exploatate decît sub sistemul de operare **MIX-PLUS.**

Fiecare task poate avea cel mult 32 Kcuvinte (8 pagini). Fiecare pagină poate avea de la 1 la 128 blocuri, un bloc avînd lungimea de 32 cuvinte.

Protecția paginilor relocate se referă la : posibilitățile de acces la pagină, direcția de acces la pagină și la lungimea de acces la pagină.

Un astfel de sistem, ce utilizează dispozitivul de relocare și protecție a memoriei, este numit *sistem mapat (sistem cu relocare).* În cazul acestor sisteme, pentru aplicații utilizator foarte mari, ce depășesc 32 kcuvinte, este posibilă extinderea dimensiunii unui task prin implementarea unei structuri de segmente rezidente în memorie, în cadrul unui spațiu de adresare logic.

Tipul de sistem, cu/sau fără relocare, afectează în mod direct modul în care sînt create task-urile utilizator.

În cazul în care sistemul este fără relocare, utilizatorii trebuie să specifice, la editarea de legături adresa de bază a partiției în care urmează să se execute un task. Task-ul obținut nu poate fi instalat într-o partiție cu o altă adresă de hază decît cea specificată la editarea de legături. Într-un astfel de sistem este necesară o reeditare de legături a task-ului pentru a putea fi încărcat și executat în altă partiție. Dimensiunea maximă a unui task este de 28 kcuvinte minus dimensiunea componentelor sistem.

În cazul în care sistemul este cu relocare, fiecare task (cu excepția task-urilor privilegiate, ce constituie o extensie a Monitorului) are o adresă de bază virtuală egală cu zero. Maparea adreselor virtuale în adrese fizice este efectuată de către unitatea de relocare și protecție a memoriei, printr-un mecanism cu totul transparent utilizatorului. În acest caz, un task poate fi încărcat și executat în orice partiție suficient de mare pentru a-l conține.

5.2. Spații de memorie în sistemul de operare MIX

Într-o memorie fizică, pe un sistem hardware cu sau fără relocare, în cadrul sistemului de operare **MIX** pot exista *patru spații de memorie*, funcțional diferite. Dimensiunea primelor trei spații este specificată la generarea sistemului de operare **MIX** și poate fi modificată dinamic la reconfigurarea sistemului, la inițializarea acestuia sau în timpul exploatării curente a sistemului.

Aceste spații de memorie sînt :

— zona Monitor ;
— zona de memorie sistem cu alocare dinamică ;
— zona task-urilor sistem și utilizator (partiții) ;
— pagina externă (de intrare/ieșire).

În sistemele fără relocare, un task poate avea acces la întreaga memorie fizică disponibilă, pe care o poate altera voit sau accidental. De aceea, e necesar să se respecte regulile de acces și distincția dintre cele patru spații de memorie ale sistemului.

În sistemele cu relocare, un task poate accesa numai memoria atașată acestuia conform drepturilor de acces, păstrîndu-se automat distincția dintre cele patru spații de memorie ale sistemului

5.2.1. Zona Monitor

Zona Monitor este plasată întotdeuna în partea inferioară a memoriei fizice și este dependentă, ca mărime, de tipul variantei de sistem alese (**MIX**, **MIX-PLUS** sau **MIX-RT**), și de serviciile Monitor selectate în procesul de generare.

La rîndul său, zona Monitor este împărțită în mai multe *subzone*, dintre care :

a) *Subzona vectorilor de întreruperi și derutări* :

— porțiunea cuprinsă între 0 și 300(8) este întotdeauna rezervată pentru vectorii standard de întreruperi și derutări (zona statică) ;

— porțiunea cuprinsă între 300(8) și 1000(8) este utilizată pentru vectorii de întreruperi ai unor configurații extinse de dispozitive de intrare/ieșire ce depășesc alocarea standard (zona flotantă).

b) *Stiva sistem* : Este utilizată pentru salvarea contextului Monitorului sau a task-urilor sistem la apariția unor întreruperi și derutări sau la apeluri interne din Monitor. Dimensiunea stivei sistem este condiționată de selectarea anumitor servicii, la generare. În cazul în care zona 300(8)÷1000(8) este liberă, subzona de stivă sistem se poate suprapune peste aceasta.

c) *Nucleul sistemului* : Conține totalitatea serviciilor și funcțiilor de control și supraveghere a activității sistemului, planificării sistem, alocatorii de resurse, tratarea unor directive Monitor, etc.

d) *Drivere pentru dispozitivele de intrare/ieșire* : Subzona poate conține drivere rezidente ale sistemului de operare **MIX-RT**. Sistemele de operare **MIX/MIX-PLUS** conțin numai drivere (sistem și utilizatoare) nerezidente în zona Monitor.

e) *Subzona de comunicație sistem* : Subzona conține tabele, liste, referințe, date comune și orice alte informații solicitate de Monitor pentru realizarea

funcțiilor sale și menținerea controlului întregului sistem. Referințele și datele comune sînt inițializate static, la generarea sistemului, și pot fi modificate dinamic, la reconfigurarea sistemului și pe parcursul exploatării acestuia. Tabelele și listele sistem au lungimi variabile, intrările fiind alocate și eliberate dinamic. În general, această subzonă este extinsă în zona de memorie sistem cu alocare dinamică.

Principalele *liste și tabele sistem* existente în această subzonă sînt :

— *Lista task-urilor instalate în sistem* (**STD**) = conține tabelele de descriere ale tuturor task-urilor (**TCB**) existente în sistem. Numărul de task-uri simultan existente în sistem este limitat de numărul de partiții disponibile în care acestea pot fi instalate. Pentru a elibera intrări **STD** corespunzătoare, taskurile instalate pot fi eliminat ela cerere (prin comanda operator **MCL/DCL REMove**) sau implicit (prin comanda operator **MCL/DCL RUN** cu încărcare, execuție și eliminare) ;

— *Lista task-urilor active* (**ATL**) = constituie o listă, ordonată după priorități, a task-urilor active și este plasată peste lista **STD.** Această listă este utilizată de planificatorii sistem pentru alocarea Unității centrale la apariția unor evenimente semnificative ;

— *Lista partițiilor din sistem* (**PCBHD**) = conține tabelele de descriere ale tuturor partițiilor (**PCB**) existente în sistem ;

— *Lista dispozitivelor de intrare-ieșire-conectate la sistem* (**DCBHD**) = conține tabelele de descriere ale tuturor dispozitivelor de intrare/ieșire conectate la sistem (**DCB**) ;

— *Lista controloarelor dispozitivelor de intrare/ieșire, conectate la sistem* (**CTBHD**) = conține tabelele de descriere ale tuturor controloarelor de dispozitive de intrare/ieșire, conectate la sistem (**CTB**) (numai sub **MIX-PLUS**) ;

— *Tabela de descriere a unui task* (**TCB**) = conține informații de identificare și stare task, referințe către alte tabele și liste sistem, capete de listă pentru diverse cozi sistem, indicatori de evenimente locali, etc. ;

— *Tabela de descriere a unei partiții* (**PCB**) = conține informații de identificare și stare partiție, referințe la alte tabele și liste sistem, etc. ;

— *Tabela de descriere a contextului unui task* (**HDR**) = conține referințe la bilioteci și zone de comun partajate, tabela numerelor logice, zone de reentranță task, zone de salvare a contextului atașat task-ului, etc. ;

— *Tabele de descriere a structurilor de date atașate unui dispozitiv de intrare/ieșire* :

— *Tabela de descriere a dispozitivelor* (**DCB**) : Există cîte un **DCB** pentru fiecare tip de dispozitiv recunoscut (conectat) la sistemul de operare **MIX**, descriind caracteristicile statice ale controlorului de dispozitiv și ale unităților atașate la acesta. În sistemul de operare **MIX-PLUS** pot exista mai multe **DCB**-uri pentru un tip de dispozitiv ;

— *Tabela de descriere a unităților atașate dispozitivelor de intrare/ieșire* (**UCB**) : Există cîte un **UCB** pentru fiecare unitate de dispozitiv (conectată la un controlor specific), descriind caracteristicile acestei unități. Tabela conține o zonă standard, cu structură identică pentru toate tipurile de dispozitiv, și o zonă specifică fiecărui tip de dispozitiv ;

— *Tabela de control a stării intrării/ieșirii* (**SCB**) : Există cîte un **SCB** pentru fiecare controlor de dispozitiv dacă acesta deservește o singură unitate de dispozitiv la un moment dat (**MIX/MIX-RT**) sau cîte un

SCB pentru fiecare unitate conectată, dacă se pot desfășura transferuri în paralel (**MIX-PLUS**) ;

— *Tabela de descriere a controloarelor de dispozitive* (**CTB**) (numai sub **MIX-PLUS**) : Fiecare **CTB** definește un tip unic de controlor în sistem, existînd cîte o tabelă pentru fiecare tip de controlor fizic.

— *Tabele de descriere a cererilor de acces la controloarele de dispozitive* (**KRB**) (numai sub **MIX-PLUS**) : Fiecare **KRB** descrie starea configurației unui controlor de dispozitiv, existînd cîte o tabelă pentru fiecare controlor fizic.

— *Tabela asignărilor logice* (**ASN**) = descriind asignările (atribuirile) unor nume logice la dispozitivele fizice conectate la sistem ;

— *Coada elementelor de ceas* (**CLQ**) = utilizate pentru sincronizarea sau replanificarea execuției task-urilor în sistem ;

— *Coada liniilor de comandă pentru procesorul de comenzi operator* **MCL** (**MCRHD**) ;

— *Coada blocurilor de cerere a întreruperilor asincrone* (**AST**), specifică pentru fiecare task din sistem ;

— *Coada pachetelor de intrare/ieșire* (**IOP**), specifică pentru fiecare driver de dispozitiv de intrare/ieșire ;

— *Alte cozi și elemente sistem ;*

— *Alte tabele diverse.*

f) *Comunuri Monitor* : Subzona conține acele directive (servicii) sistem ce pot fi scoase în afara Nucleului pentru mărirea zonei primare de memorie sistem cu alocare dinamică (**DSR**).

Comunurile Monitor (6 în prezent) pot exista numai pe sistemele cu relocare și se pot aloca oriunde în memoria fizică, în afara Nucleului, folosind pentru mapare **APR**-ul 5 din spațiul de instrucțiuni (**I**).

În cazul sistemelor de operare **MIX-PLUS** de pe calculatoarele cu spații **I** și **D** separate, zona de comun se mapează și cu **APR**-ul 5 din spațiul de date (**D**).

Denumirea comunurilor Monitor este :

— **EXCOM1/EXCOM2** = pentru sistemele **MIX/MIX-RT** ;

— **DR1MIX/DR2MIX** = pentru sistemul **MIX-PLUS V1.0** ;

— **DR1MIX/DR2MIX/DR3MIX/DR4MIX/VECMIX/DCMMIX** = pentru sistemul **MIX-PLUS V2.0.**

5.2.2. Zona de memorie sistem cu alocare dinamică

În executarea anumitor servicii Monitor sau sistem, cît și pentru extinderea dinamică a numărului de elemente din listele și tabelele sistem, este necesară o zonă de memorie sistem suplimentară.

Alocarea și eliberarea elementelor sistem obținute din această zonă se face în mod dinamic, sub controlul direct al Monitorului **MIX**. De obicei, în cazul indisponibilității unei zone de alocare dinamică, Monitorul informează task-ul ce a solicitat, indirect prin intermediul serviciilor sistem, o astfel de alocare. Task-ul utilizator poate aștepta apariția unei disponobilități de memorie în zona de alocare dinamică și poate repeta serviciul sistem solicitat.

Dimensiunea zonei de memorie sistem cu alocare dinamică reprezintă un parametru important al sistemului de operare **MIX.** O dimensiune prea mică poate duce la blocări frecvente ale task-urilor în așteptarea eliberării de memorie dinamică ; o dimensiune prea mare poate constitui o risipă de spațiu în detrimentul zonei de memorie alocate partițiilor din sistem.

Zona de memorie sistem cu alocare dinamică se poate împărți, logic și fizic, în una **(MIX-RT)**, două **(MIX/MIX-PLUS)** sau patru **(MIX-PLUS)** subzone distincte :

— o zonă primară cu alocare dinamică **(DSR)**, numită și *„pool"*, continuă cu zona Monitor ;

— o zonă secundară cu alocare dinamică **(SECPOL)**, numită și *„pool secundar"*, în afara zonei Monitor (existentă numai pe sistemele **MIX/MIX-PLUS) ;**

— o extensie a zonei primare cu alocare dinamică, în spațiul de date **(DSPMIX)**, numită și *„pool mapat în spațiul* **D**", în afara zonei Monitor (existentă numai pe sistemele **MIX-PLUS** de pe calculatoarele cu spații **I** și **D** separate) ;

— o zonă terțiară cu alocare dinamică **(CPUPOL)**, numită și *„pool multiprocesor"* (pentru configurațiile hardware cu mai multe Unități Centrale), în afara zonei Monitor (existentă numai pe sistemele **MIX-PLUS** de pe configurații multiprocesor).

Zona Monitor (fără comunuri) și zona primară de memorie sistem cu alocare dinamică **(DSR)** sînt protejate, pe sistemele mapate, printr-un set special de registre de pagină corespunzătoare *modului Sistem (Kernel)* și spațiului de instrucțiuni **(I).** Cele două zone sînt cuprinse între limitele de memorie 0 și 16—20 Kcuvinte (4—5 **APR**-uri), în funcție de varianta de sistem aleasă și dimensiunea zonei primare de memorie sistem cu alocare dinamică.

Dimensiunea zonei **DSR,** specificată la generarea sistemului, poate fi de obicei modificată dinamic (în sensul extinderii) la reconfigurarea sau inițializarea primară a sistemului (comanda **VMR** sau operator **MCL/DCL SET/POOL).** Această observație este valabilă în special pentru sistemul **MIX-RT** și mai puțin pentru sistemele **MIX/MIX-PLUS,** întrucît acestea au Monitorul extins la limita superioară (20Kc) și o dimensiune medie a zonei **DSR** de cca 8 kocteți.

În cazul sistemelor **MIX-PLUS,** de pe calculatoarele cu spații **I** și **D** separate, zona primară cu alocare dinamică **(DSR)** urmează după subzona vectorilor de întreruperi/derutări și stiva sistem. Suma acestora nu poate depăși 4 kcuvinte (1 **APR)** și zona respectivă se mapează atît în spațiul **I,** cît și în spațiul **D.**

Zona secundară cu alocare dinamică **(SECPOL),** se creează în afara Monitorului, într-o partiție specifică a cărei dimensiune și plasare în spațiul virtual se poate modifica dinamic la reconfigurarea sistemului. Sub **MIX-PLUS,** zona se poate extinde dinamic și prin comanda operator **SEC.**

Dimensiunea acestei zone este de cca. 4 Kocteți pe sistemul **MIX** și de min. 32—64 kocteți pe sistemul **MIX-PLUS.**

Spre deosebire de sistemul **MIX** în care zona **SECPOL** este utilizată numai pentru transmiterea de mesaje (de lungime fixă sau variabilă) între task-uri, sistemul **MIX-PLUS** utilizează intensiv această zonă eliminînd din zona primară **(DSR)** majoritatea structurilor dinamice mari consumatoare de memorie : **TCB**-urile prototip, liniile de comandă **MCL/DCL/CLI,** pachetele

de eroare, informațiile de contabilizare, extensiile **UCB** pentru driver-ul de terminal, etc.

Maparea acestor structuri se face prin intermediul **APR**-ului 5 din spațiul de instrucțiuni (**I**). În cazul sistemelor de operare **MIX-PLUS** de pe calculatoarele cu spații **I** și **D** separate, structurile din zona **SECPOL** se mapează și cu **APR**-ul 5 din spațiul de date (**D**).

Pe sistemele **MIX-PLUS** de pe calculatoarele cu spații I și **D** separate, zona primară cu alocare dinamică (**DSR**) se extinde în afara zonei Monitor, putînd avea maximum 24 Kcuvinte lungime. Această zonă (**DSPMIX**) este mapată cu **APR**-urile 1 la 6 din spațiul de date (**D**).

În cadrul familiei de sisteme de operare **MIX**, solicitările privind zona sistem cu alocare dinamică („*pool*") depind de configurație, aplicație și de gradul de încărcare a sistemului. Pentru satisfacerea unor situații „de vîrf" (de supraîncărcare a sistemului) este necesar suficient „*pool*" ; în caz contrar, are loc o degradare a performanțelor generale ale sistemului de operare.

Întrucît majoritatea funcțiilor sistem necesită „*pool*", acesta se poate epuiza la o încărcare mare a sistemului : prea multe taskuri instalate, prea multe volume montate, etc. În acest caz nu se produce o cădere de sistem, ci o funcționare aparent anormală a sistemului.

Pentru prevenirea situațiilor de epuizare, în familia de sisteme de operare **MIX** (mapate), are loc monitorizarea consumului de memorie primară cu alocare dinamică (**DSR**), efectuată de către Monitor și un *task sistem special* (**PMT**). Limitele de monitorizare și dimensiunile critice de alarmă se stabilesc static, la reconfigurare, sau dinamic, prin comanda operator **SET/PLCTL**....

Monitorul, pe baza limitelor stabilite, supervizează dimensiunea totală și fragmentarea zonei primare cu alocare dinamică (**DSR**), detectează evenimentele majore asociate epuizării zonei dinamice și pasează controlul task-ului **PMT**. Acesta, pe baza informațiilor specificate de Monitor, execută acțiuni specifice condiției detectate : detectarea limitelor inferioare de epuizare a zonei de „*pool*", prevenirea utilizatorilor neprivilegiați de a nu mai intra în sistem, suprimarea secvenței **INSTALL/RUN/REMOVE** pe terminalele neprivilegiate, emiterea unor mesaje de avertizare pe toate terminalele legate în sistem, afișarea pe consola operator a unor statistici privind dimensiunea și fragmentarea zonei dinamice, etc.

5.2.3. Zona task-urilor sistem și utilizator

Zona de memorie disponibilă după Monitor, este împărțită în cadrul sistemului de operare **MIX** în partiții, în care se pot executa task-uri sistem sau utilizator.

Zona partițiilor se întinde de la limita superioară a Monitorului pînă la 28 Kcuvinte (cazul sistemelor fără relocare) sau maximum 2048 Kcuvinte (cazul sistemelor cu relocare).

Numărul partițiilor este limitat numai de memoria fizică disponibilă la un moment dat.

5.2.4. Pagina externă (de intrare/ieșire)

Arhitectura minicalculatoarelor românești rezervă ultimele 4 cuvinte de memorie pentru registrele de stare și control ale dispozitivelor de intrare/ieșire.

Pe sistemele fără relocare, această zonă, cuprinsă între 28 și 32 kcuvinte, este proiectată automat (prin hardware) în zona 124 ÷ 128 kcuvinte, rezervînd sistemului de operare (inclusiv task-urile utilizator) un spațiu de lucru de numai 28 kcuvinte.

În sistemele de relocare, memoria disponibilă utilizatorilor este de maximum 124 kcuvinte (pe minicalculatoarele **I-100, I-102F, CORAL 4011(A)**), sau maximum 1 920 kcuvinte (pe minicalculatoarele **I-102F** cu extensie de memorie, **I-102/4M, I-106, CESAR-16**) sau maximum 2 044 kcuvinte (pe minicalculatoarele **CORAL 4030, 4021, 4015, I-1016**).

Această zonă este numită *pagina externă* sau *pagina de intrare/ieșire*.

În sistemele cu relocare, la ea au acces Monitorul și toate task-urile privilegiate. În sistemele fără relocare, toate task-urile au acces la pagina externă.

Task-urile neprivilegiate nu au acces în pagina externă decît prin intermediul unor partiții de comun mapate în pagina externă.

În fig. 5.1 este prezentat un exemplu de alocare a spațiilor de memorie în sistemul de operare **MIX-RT**, pentru o configurație medie, de 32 kcuvinte. Figura 5.2 prezintă un exemplu de alocare similar pentru sistemul de operare **MIX**.

5.3. Partiții și subpartiții

Un task se execută într-o zonă continuă de memorie numită *partiție*. Fiecare partiție se caracterizează prin următoarele 4 entități :

— *Nume* : De la 1 la 6 caractere, în care primul este alfabetic ;

— *Adresă de bază* : Adresa de la care începe partiția (multiplu de 32 cuvinte) ;

— *Dimensiune* : Numărul de cuvinte conținut în partiție (multiplu de 32 cuvinte) ;

— *Tip* : Partiție principală (**MAIN**) sau subpartiție (**SUB**).

Relația între task și partiția în care se execută depinde de tipul sistemului de operare (cu sau fără relocare). Într-un sistem fără relocare, un task poate fi executat numai într-o partiție avînd aceeași adresă de start cu cea specificată la editarea de legături. Într-un sistem cu relocare, un task poate fi executat în orice partiție suficient de mare pentru a-l conține.

5.3.1. Tipuri de partiții

Sistemele de operare **MIX/MIX-RT** suportă trei tipuri de partiții :

— partiții controlate de utilizator ;
— partiții controlate de sistem ;
— partiții de comun :
  — banale ;
  — proiectate în pagina externă.

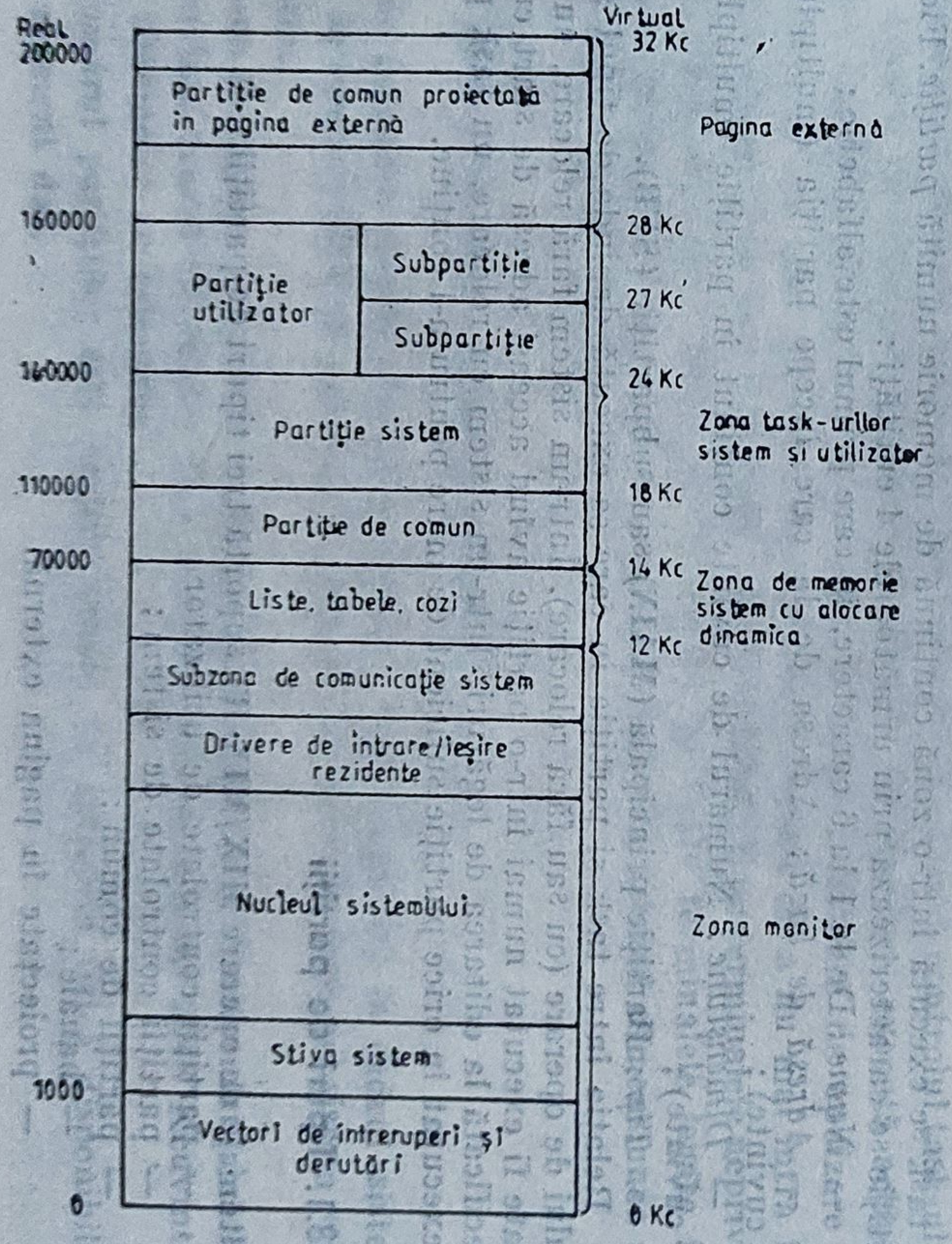


Fig. 5.1. Spațiile de memorie din cadrul sistemului **MIX-RT** (exemplu)

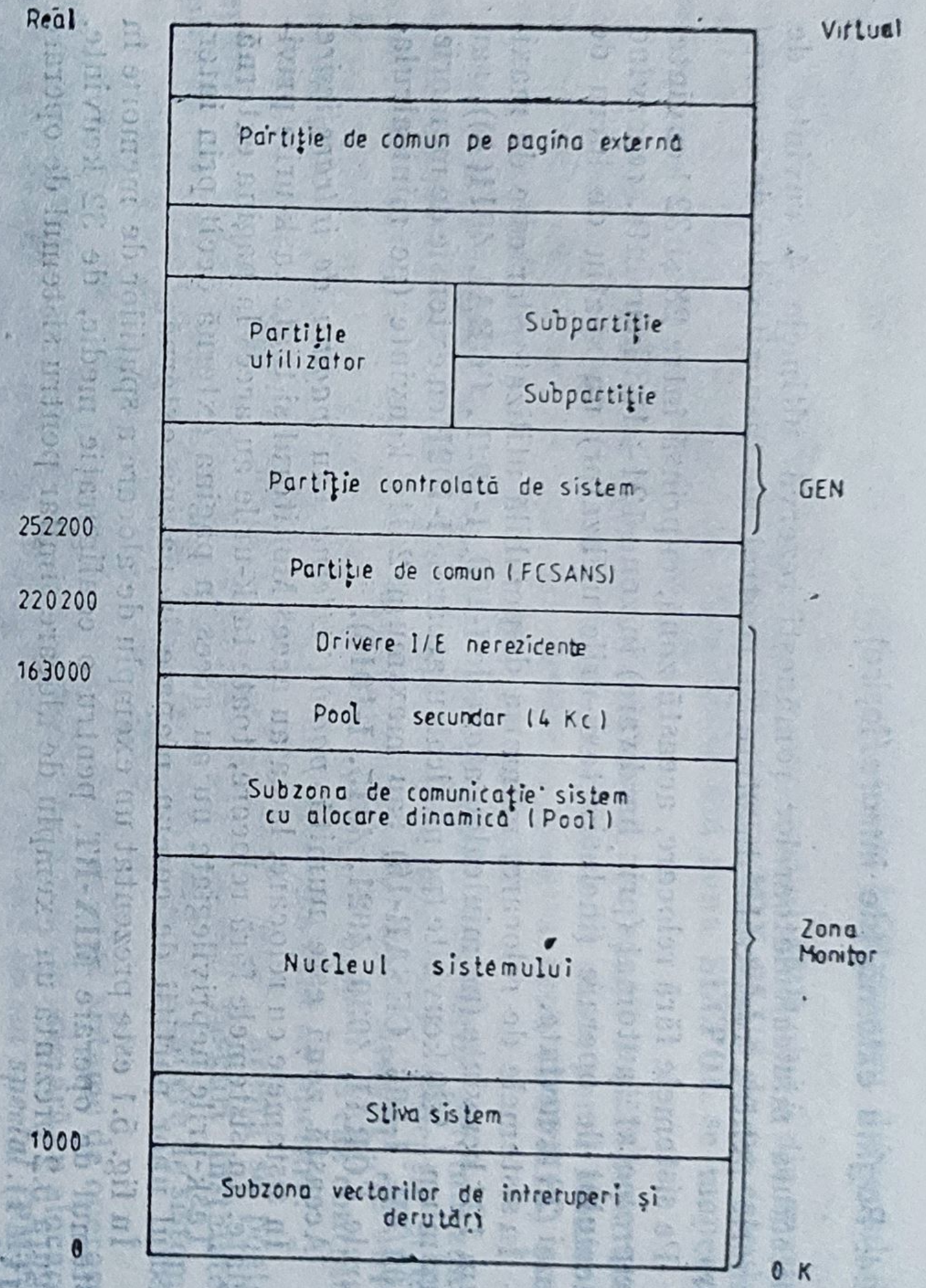


Fig. 5.2. Spațiile de memorie în sistemul **MIX**

Sistemul de operare **MIX-PLUS** suportă un singur tip de partiţii, partiţ controlate de sistem. Încărcarea bibliotecilor şi blocurilor de comun rezidente are loc în partiţiile controlate de sistem.

Partiţiile controlate de utilizator şi sistem sînt alocate pentru execuţia task-urilor sistem şi utilizator. Partiţiile de comun banale conţin biblioteci şi blocuri de comun rezidente. Partiţiile de comun proiectate în pagina externă sînt utilizate de task-urile neprivilegiate pentru accesarea registrelor de control şi stare ale dispozitivelor periferice (registre plasate în pagina externă).

Numărul partiţiilor şi caracteristicile acestora sînt stabilite *static* (la generare) sau *dinamic* (la reconfigurarea sistemului sau în timpul exploatării curente a acestuia, de către operator).

Într-un sistem de operare **MIX** se poate crea orice număr de partiţii, de orice tip, cu condiţia să existe suficientă memorie disponibilă. Alocarea unei partiţii de orice tip se face cu comanda operator **SET/MAIN . . .(MIX/MIX-RT)** sau **SET/PAR . . . (MIX-PLUS)**, iar eliberarea spaţiului de memorie alocat acesteia prin comanda complementară **SET/NOMAIN . . .(MIX/MIX-RT)** sau **SET/NOPAR . . .(MIX-PLUS)**.

5.3.2. Partiţii controlate de utilizator

O astfel de partiţie poate conţine un sigur task la un moment dat. Utilizatorul are un control deplin asupra partiţiei, aceasta constituind o soluţie ideală pentru aplicaţiile în timp real. Acest tip de partiţie poate fi implementat atît în sistemele fără relocare cît şi în cele cu relocare.

O partiţie controlată de utilizator poate fi împărţită în una sau mai multe subpartiţii.

5.3.3. Subpartiţii

O partiţie controlată de utilizator poate fi împărţită în maximum 7 subpartiţii, fără suprapunere. O subpartiţie poate conţine un singur task la un moment dat. Întrucît subpartiţia ocupă acelaşi spaţiu fizic cu partiţia principală, nu pot exista task-uri simultan rezidente în partiţia principală şi în una sau mai multe subpartiţii. Întrucît subpartiţiile nu se suprapun între ele, într-o partiţie principală vidă se pot executa în paralel un număr de 7 task-uri.

Scopul subpartiţionării este acela de utilizare eficientă a spaţiului de memorie în sistemele fără relocare. De exemplu, într-un sistem **MIX-RT**, coexistă în subpartiţii un număr de task-uri de timp real de dimensiuni mici. În momentul terminării activităţii acestora, spaţiul de memorie ocupat de partiţia principală poate fi alocat unui task de dezvoltare de programe, de dimensiuni mari.

Alocarea unei subpartiţii într-o partiţie principală se face prin comanda operator **SET/SUB . . .**, iar eliberarea spaţiului alocat prin comanda complementară **SET/NOSUB . . .**.

Un exemplu de divizare a unei partiţii principale în mai multe subpartiţii este prezentat în fig. 5.3. Partiţia principală, numită **GEN**, cu o lungime de 8 kcuvinte, este împărţită în a subpartiţii, **INIT** de 1 kcuvinte, **TEST** de 4 kcuvinte şi **SAVPAR** de 3 kcuvinte.

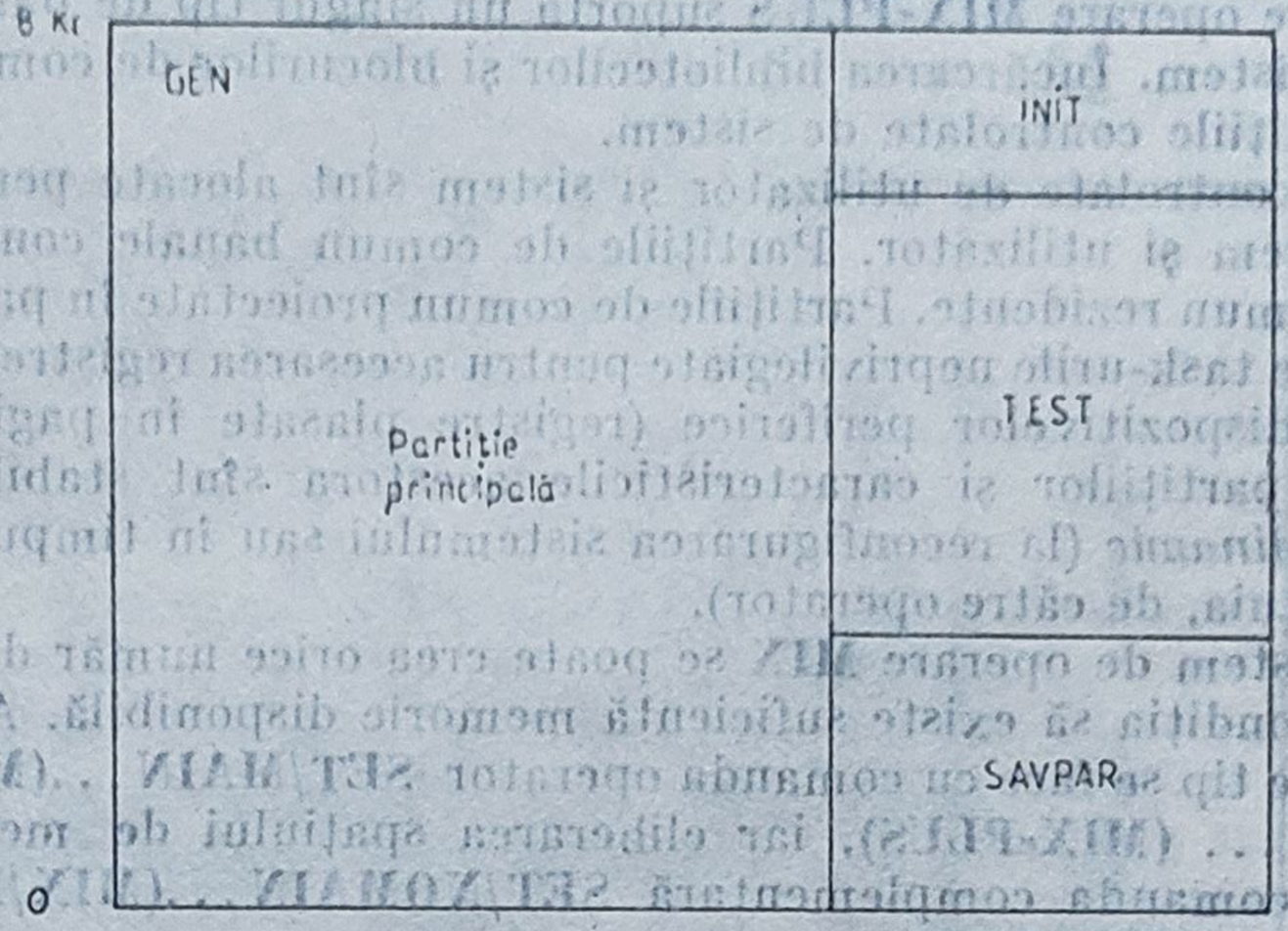


Fig. 5.3. Împărţirea unei partiţii principale în mai multe subspaţii

5.3.4. Partiţii controlate de sistem

O astfel de partiţie poate conţine unul sau mai multe task-uri la un moment dat, plasarea task-urilor în memorie fiind controlată de sistem. Monitorul gestionează spaţiul disponibil dintr-o astfel de partiţie încercînd să-l folosească, pe cît posibil, pe tot. Alocarea implică şi o compactare (rearanjare) posibilă a task-urilor rezidente pentru a face loc altora.

În cazul în care mai multe task-uri active solicită acces la (aşteaptă să fie încărcate într-o) partiţie controlată de sistem, Monitorul alocă memoria dinamică din partiţie conform unui algoritm specific tipului de sistem de operare (**MIX** sau **MIX-PLUS**), cu următoarele faze :

— utilizează nivelul de prioritate ataşat task-ului aflat în aşteptare (1 la 250) pentru a determina accesibilitatea acestuia la partiţia controlată de sistem ;

— caută, începînd cu baza partiţiei, o zonă de memorie liberă, contiguă, suficient de mare pentru a conţine task-ul aflat în aşteptare ;

— creează o subpartiţie dinamică în cadrul partiţiei controlate de sistem ;

— încarcă tesk-ul aflat în aşteptare în zona de memorie descrisă de subpartiţia dinamică (operaţiune efectuată cu ajutorul unui task sistem special **LDR**) ;

— elimină subpartiţia şi eliberează spaţiul în partiţia controlată de sistem, la terminarea execuţiei task-ului.

În cazul în care nu există suficient spaţiu pentru încărcarea task-ului aflat în aşteptare, Monitorul activează mecanismul de evacuare pentru a crea spaţiu liber în partiţia controlată de sistem.

În sistemul de operare **MIX**, contorizarea intrărilor/ieşirilor neterminate se face „per task", Monitorul neputînd determina uşor căror partiţii le sînt asociate aceste intrări/ieşiri. Din acest motiv, se pot evacua numai subpartiţiile ce conţin regiuni de tip task, partiţiile de comun şi regiunile de comun dinamice (create cu ajutorul directivelor de gestiune a memoriei) neputînd fi evacuate.

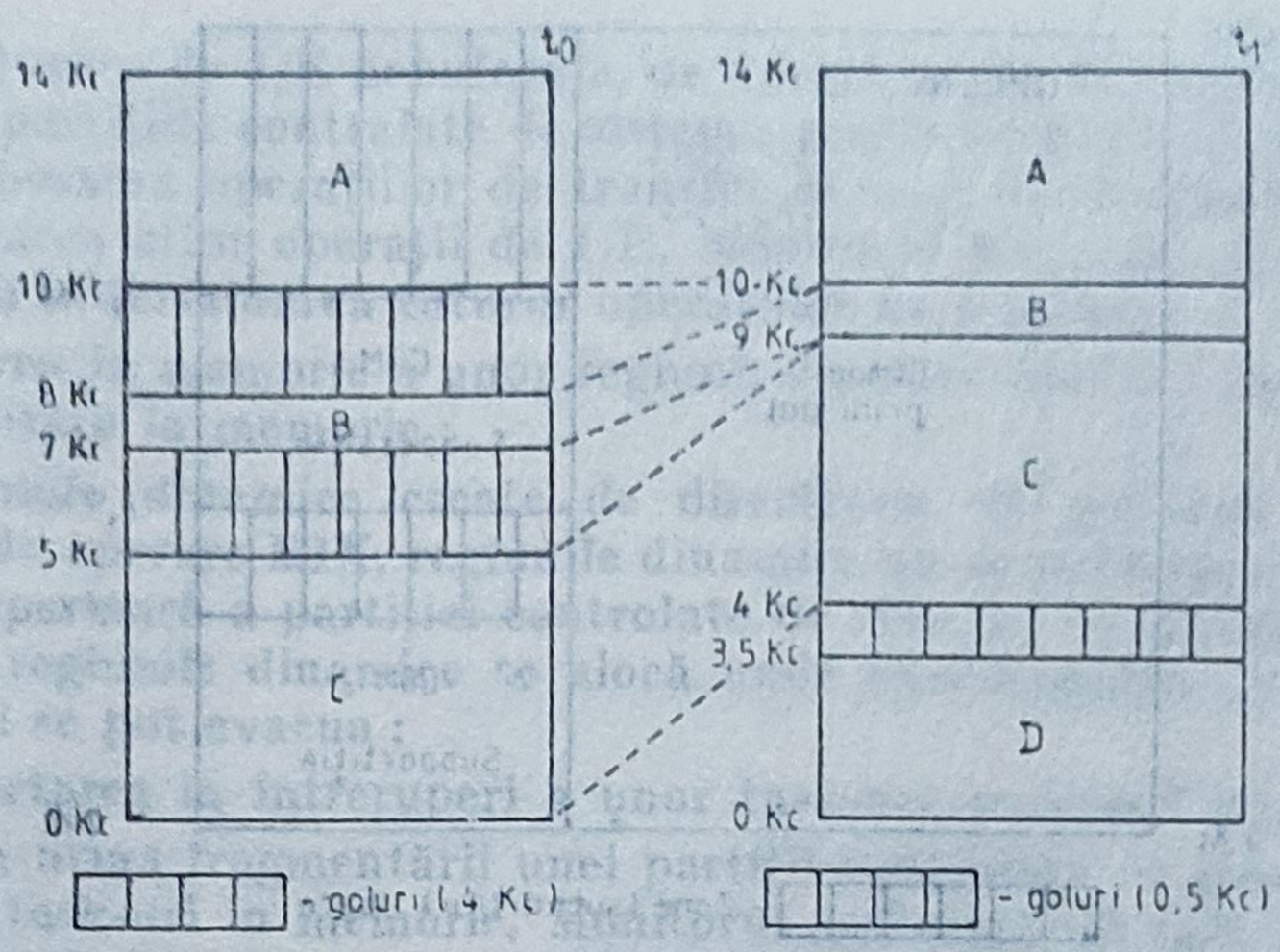


Fig. 5.4. Instalarea task-urilor într-o partiție controlată de sistem

În sistemul de operare **MIX-PLUS**, contorizarea intrărilor/ieșirilor neterminate se face „per partiție", Monitorul putînd evacua, dacă nu există I/E în așteptare, atît subpartițiile ce conțin regiuni de tip task cît și subpartițiile de comun (statice sau dinamice).

În figura 5.4 este prezentată harta de alocare a unei partiții controlate de sistem la două momente de timp. La momentul t_0, în partiție erau instalate 3 task-uri, **A**, **B** și **C**, cu goluri între ele. La momentul t_1, în urma unei rearanjări *(compactări)*, în spațiul disponibil poate fi instalat și task-ul **D**.

Partițiile controlate de sistem pot fi definite și utilizate numai în sistemele cu relocarea memoriei.

Aceste partiții sînt destinate pentru executarea unor task-uri ce nu au caracteristici de timp real (dezvoltare de programe, etc.).

5.3.5. Partiții de comun banale

Partițiile de comun banale conțin biblioteci de rutine partajate sau blocuri de comun, rezidente.

Bibliotecile de rutine pot fi partajate între mai multe task-uri și au acces, de obicei, numai în citire.

Blocurile de comun reprezintă zone de comunicație între mai multe task-uri, cu acces în citire/scriere sau numai în citire.

O partiție de comun banală este considerată de Monitor ca o partiție de tip utilizator, putînd fi subpartiționată în scopul utilizării parțiale a acesteia de către mai multe task-uri utilizator.

Figura 5.5 reprezintă o partiție de comun partajată de 3 utilizatori: unul principal **COMGEN**, cu acces în citire/scriere la întreaga partiție și doi secundari, **COM1** și **COM2**, cu acces în citire la două subzone ale comunului principal (subpartiții).

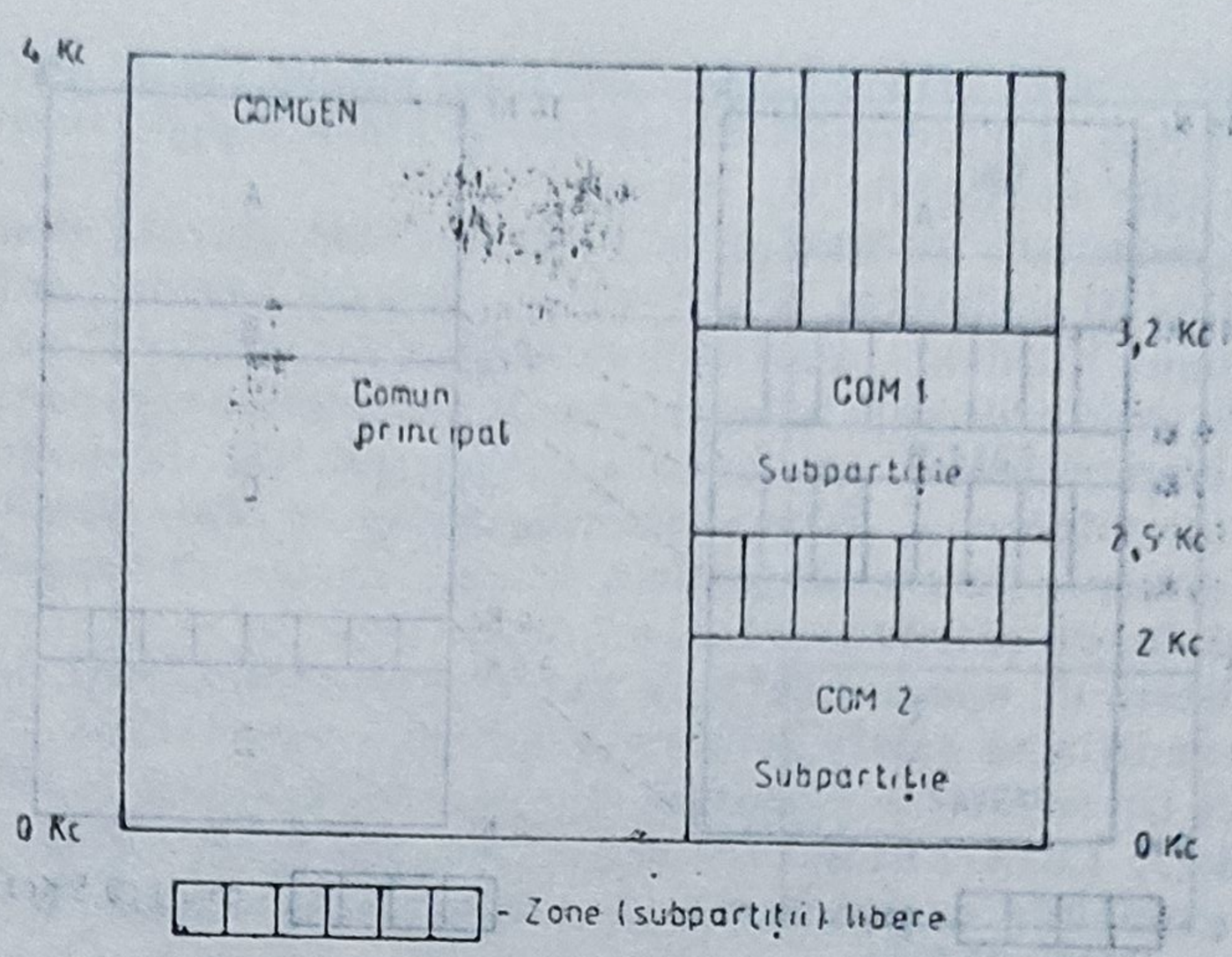


Fig. 5.5. Partajarea unei zone de comun

5.3.6. Partiții de comun proiectate în pagina externă

În general, pagina externă (de intrare/ieșire) poate fi accesată, fără nici o restricție, numai de task-urile privilegiate (sisteme cu sau fără relocare). În acest caz, accesul la pagina externă se face prin adrese absolute.

În cazul în care se dorește un acces restrictiv la pagina externă, task-urile se pot lega la o partiție de comun proiectată în această zonă.

O partiție de comun proiectată în pagina externă are adresele fizice limitate de un set de locații din această pagină.

Aceste partiții sînt utile :

— task-urilor neprivilegiate din sistemele cu relocare pentru a accesa direct registrele de control și stare ale dispozitivelor periferice plasate în pagina externă ;

— limitării accesului la pagina externă prin specificarea unor zone precise în cadrul blocului de comun.

Această soluție este extrem de utilă task-urilor de timp real ce au o interfață nestandard cu anumite dispozitive de intrare/ieșire, lucrînd direct la nivelul registrelor de interfață.

O partiție de comun proiectată în pagina externă nu poate fi subîmpărțită în mai multe subzone.

5.4. Compactarea memoriei în partiții sistem

Fragmentarea spațiului, care apare într-o partiție controlată de sistem, are multiple cauze, dintre care se pot aminti :

— încărcarea driver-elor de dispozitiv în mijlocul partiției controlate de sistem ; fragmentarea se elimină prin încărcarea driver-elor în partea superioară a partiției — comanda operator **LOA .../HIGH.**

— efectuarea de I/E nebuferate, de durată mare, în corpul unor task-uri instalate în partițiile controlate de sistem : aceste task-uri nu pot fi evacuate pînă la terminarea operațiilor de transfer în curs de desfășurare ; pentru a preveni lansarea altor operații de I/E, Monitorul blochează execuția acestor task-uri pînă la terminarea tuturor operațiilor de transfer ;

— fixarea în memorie a unor regiuni, de către Monitor, la apariția unor erori de paritate la memorie ;

— regiunile dinamice create de directivele de gestiune a memoriei ; în sistemul de operare **MIX**, regiunile dinamice nu se pot evacua, dar se alocă în partea superioară a partiției controlate de sistem ; în sistemul de operare **MIX-PLUS**, regiunile dinamice se alocă unde există spațiu (fragment) liber în partiție și se pot evacua ;

— conectarea la întreruperi a unor task-uri, utilizînd directiva **CINT$**.

Dacă în urma fragmentării unei partiții controlate de sistem nu se mai pot încărca task-uri în memorie, Monitorul activează un task sistem, numit „*shuffler*", care implementează o serie de algoritmi de compactare a memoriei fragmentate. Efectuînd mai multe treceri prin lista subpartițiilor atașate partiției controlate de sistem, task-ul de compactare schimbă fizic poziția subpartițiilor atașate unor task-uri (și eventual evacuează o serie de task-uri), în încercarea de alocare a unui spațiu liber suficient pentru încărcarea unui task de prioritate mai mare, aflat în așteptare pentru resursa memorie.

5.5. Spații de adresare

5.5.1. Posibilitățile de adresare ale unui task

În sistemul de operare **MIX**, un task specifică o adresă printr-un cuvînt de 16 biți. Cea mai mare adresă exprimată printr-un astfel de cuvînt este de 65 536 octeți sau 32 768 cuvinte (în limbajul comun, 32 kcuvinte). Un task poate da aceea accesa în mod direct cel mult 32 kcuvinte.

Pentru a depăși această limită, un task poate fi împărțit în *segmente* : un singur segment-rădăcină, prezent tot timpul în memorie și un număr variabil de segmente ce pot fi aduse rapid în memorie de pe disc la cerere (încărcare manuală) sau în momentul unei referiri la segment (încărcare automată). Cu toate acestea, dimensiunea combinată a diferitelor segmente prezente la un moment dat în memorie, nu poate depăși 32 kcuvinte.

O astfel de structură este accesibilă numai unui sistem de operare cu disc cum este de pildă **MIX/MIX-PLUS.**

Sistemul de operare **MIX** oferă utilizatorului un set de directive sistem (numite directive de gestiune a memoriei) ce permit depășirea restricției de adresare a 32 kcuvinte prin schimbarea dinamică a referirilor la locațiile cuprinse într-o gamă de adrese.

Implementarea și utilizarea directivelor de gestiune a memoriei necesită prezența Unității de relocare și protecție a memoriei în construcția Unității centrale. Fără acest echipament hardware nu poate fi vorba de o depășire a limitei de 32 kcuvinte.

Utilizînd aceste directive, un task poate fi conceput ca avînd o *structură segmentală* (rădăcină și un număr variabil de segmente), rezidente însă total în memorie. Viteza de execuție a task-ului crește, durata transferurilor de intrare/ieșire se reduce la zero, pe seama unui consum suplimentar de memorie.

Pentru a rezolva referințele dintre segmente, task-ul își proiectează adresele sale virtuale la diferite adrese fizice, în diferite zone logice aparținînd task-ului. Utilizînd directivele de gestiune a memoriei, această proiectare poate fi făcută în mod automat sau la un nivel vizibil și controlat de către utilizator.

5.5.2. Spații de adresare ale sistemului MIX

În sistemul de operare **MIX** se definesc *trei tipuri de spații de adresare*:

1) *Spațiul de adresare fizic*: Acesta constă din memoria fizică în care task-ul este încărcat și se poate executa;

2) *Spațiul de adresare logic*: Acesta reprezintă întregul spațiu de adresare fizic la care task-ul are drepturi de acces; un task poate împărți spațiul său de adresare logic în mai multe zone, numite regiuni; fiecare regiune reprezintă un bloc contiguu de memorie;

3) *Spațiul de adresare virtual*: Acesta corespunde celor 32 kcuvinte de adrese pe care un task îl poate specifica în mod explicit utilizînd cuvinte de 16 biți; un task poate împărți spațiul său de adresare virtual în mai multe segmente numite *segmente* (*sau ferestre de adrese*) *virtuale*.

În cazul în care un task nu utilizează directivele de gestiune a memoriei, spațiile sale de adresare logic și virtual corespund în mod direct; o adresă virtuală va referi întotdeuna o aceeași locație logică, ambele tipuri avînd o dimensiune maximă de 32 kcuvinte.

În schimb, utilizînd directivele de gestiune a memoriei, un task poate asigna sau proiecta o gamă de *adrese virtuale* (segment virtual) la diferite *zone logice* (regiuni), permițînd utilizatorului extinderea spațiului de adresare logic atașat acestuia peste limita de 32 kcuvinte.

De remarcat că, în sistemul de operare **MIX-PLUS**, un task obișnuit poate conține 32 Kcuvinte de spațiu virtual și poate accesa cca. 32 Kcuvinte de spațiu fizic. Un task utilizînd însă spații de instrucțiuni și date poate conține 64 Kcuvinte de spațiu virtual și poate accesa cca. 64 Kcuvinte de spațiu fizic. În acest caz, utilizarea directivelor de gestiune a memoriei extinde spațiul logic de adresare peste limita de 64 Kcuvinte.

5.6. Segmente virtuale

În scopul facilitării proiectării adreselor virtuale pe diferite zone logice, utilizatorul trebuie să-și împartă spațiul de adresare virtual (32 kcuvinte) în segmente. Aceste segmente, numite în cadrul sistemului **MIX** *segmente*

sau *ferestre de adrese virtuale (address windows)* cuprind o zonă contiguă de adrese virtuale şi sînt aliniate la o adresă multiplu de 4 kcuvinte.

Numărul de segmente virtuale definit de un task poate varia de la 1 la 7 (**MIX**) sau de la 1 la 23 (**MIX-PLUS**), dimensiunea fiecăruia fiind de la minimum 32 cuvinte la maximum 32 K (—32) cuvinte. Fiecărui segment virtual definit îi corespunde un bloc de descriere a segmentului virtual (**WDB**), plasat în antetul de task. Identificarea segmentelor virtuale se face printr-un număr de la 0 la 7 (**MIX**) sau de la 0 la 23 (**MIX-PLUS**), ce reprezintă şi un index în tabela blocurilor de descriere a segmentelor virtuale aparţinînd unui task.

În urma oricărei editări de legături a unui task se creează cel puţin un segment virtual (cu numărul 0) ce identifică antetul şi segmentul rădăcină aparţinînd task-ului. Proiecţia acestui segment pe spaţiul logic ataşat task-ului se face în mod automat de către Monitor, în momentul încărcării task-ului în memorie. Pentru task-urile cu spaţii I şi D separate (**MIX-PLUS**), se creează cel puţin două segmente virtuale (0 — pentru spaţiul I şi 1 — pentru spaţiul D).

Blocurile de descriere a segmentelor virtuale plasate în antetul de task (header) sînt iniţializate la instalare, Monitorul utilizînd informaţiile de mapare continute în aceste blocuri pentru setarea registrelor de alocare (**APR**) ale unităţii de relocare şi protecţie a memoriei. Maparea se poate schimba, dinamic, utilizînd directivele de gestiune a memoriei.

Directivele de gestiune a memoriei permit următcarele acţiuni utilizator asupra segmentelor virtuale :

— crearea unui nou segment virtual şi opţional, proiectarea acestuia într-o regiune (**CRAW$**) ;

— eliminarea unui segment virtual cu ştergerea eventuală a proiecţiei acestuia dintr-o regiune (**ELAW$**) ;

— proiectarea unui segment virtual într-o regiune (**MAP$**) ;

— ştergerea proiecţiei unui segment virtual dintr-o regiune (**UMAP$**) ;

— obţinerea contextului de proiectare al unui segment virtual într-o regiune (**GMCX$**).

În sistemul de operare **MIX-PLUS** este posibilă creerea şi maparea unui segment virtual şi în spaţiul I *Supervizor* sau spaţiul D *Utilităţilor* (conform opţiunilor de descriere a **WDB**).

În cazul utilizării directivelor de gestiune a memoriei, relaţia dintre spaţiile de adresare logic şi virtual aparţinînd unui task trebuie văzută în contextul existenţei segmentelor virtuale şi a regiunilor. Atîta vreme cît o adresă virtuală face parte dintr-un segment virtual, ea poate fi bine precizată. În mod similar, un segment virtual poate fi proiectat numai într-o zonă egală sau mai mare cu o regiune existentă în cadrul spaţiului de adresare logic ataşat task-ului

Corespondenţa segmente virtuale — spaţiu de adresare virtual este prezentată schematic în fig. 5.6. În acest exemplu, spaţiul virtual de 32 kcuvinte este divizat în trei segmente virtuale (0, 1 şi 2), de dimensiuni arbitrare. Zonele haşurate indică porţiuni ale spaţiului de adresare neutilizate.

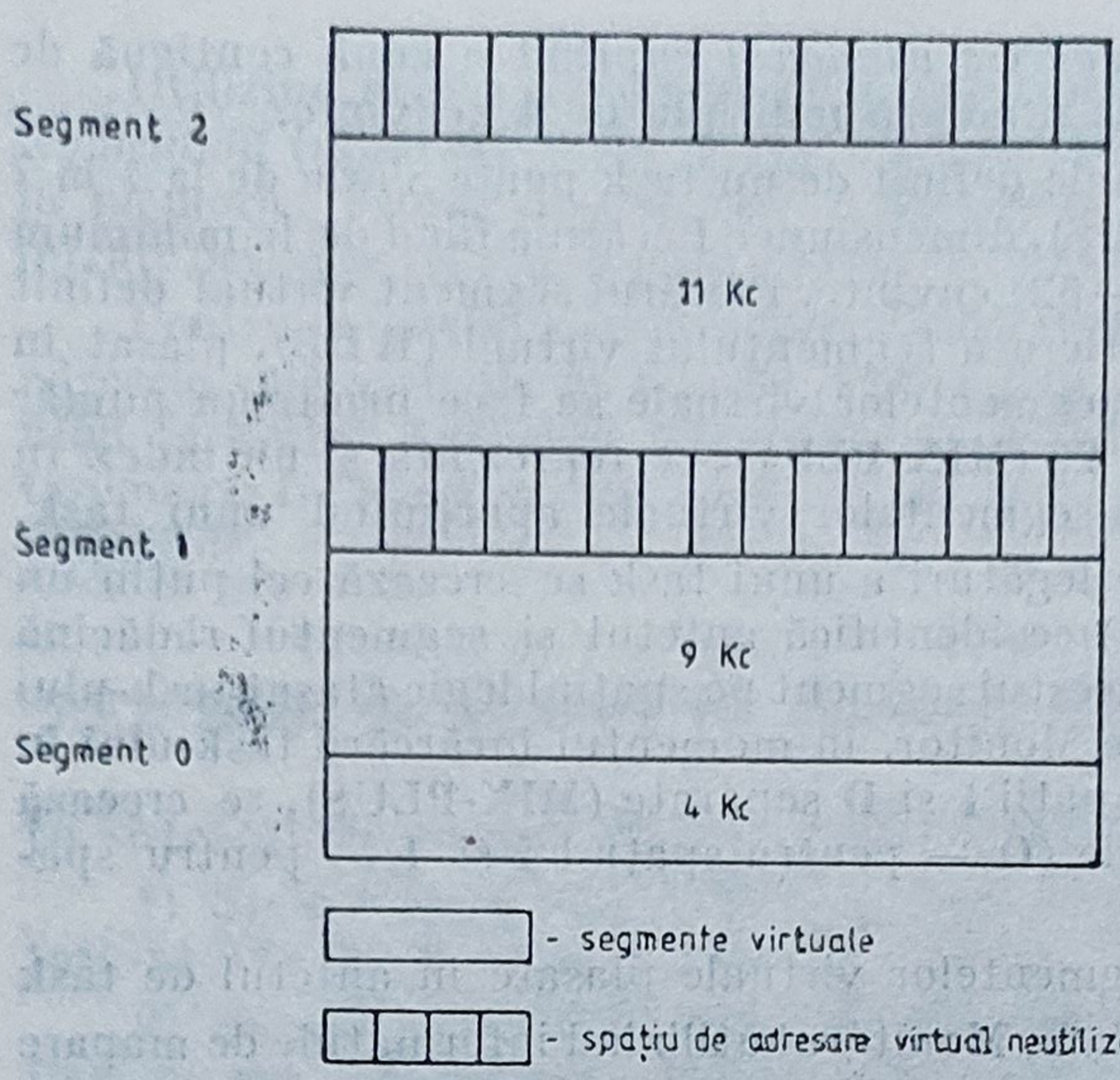


Fig. 5.6. Reprezentarea unui task prin segmente virtuale

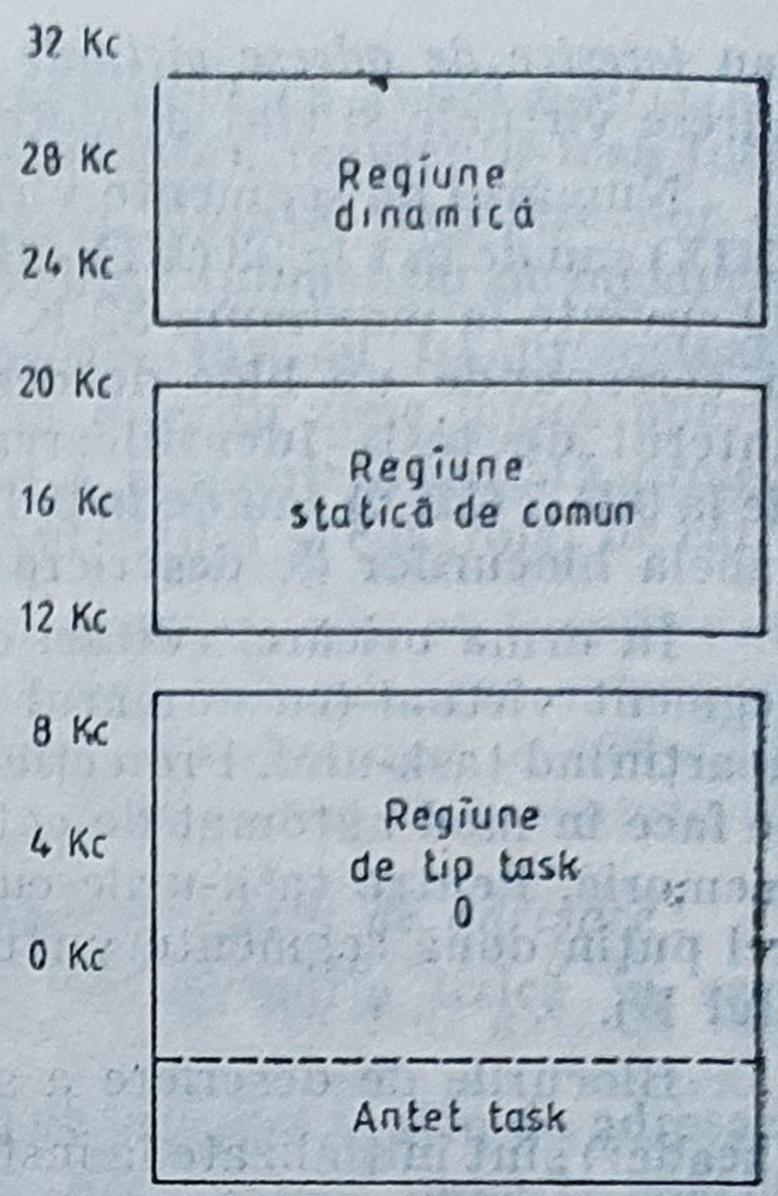


Fig. 5.7. Spațiul de adresare logic atașat unui task

5.7. Regiuni

Contextul de proiecții segmente virtuale — regiuni determină partea din spațiul de adresare logic la care un task are drepturi de accesla un moment dat.

Spațiul de adresare logic este format din mai multe tipuri de regiuni :

1) *Regiune de tip task* : această regiune reprezintă o zonă contiguă de memorie în care task-ul este încărcat și se poate executa ;

2) *Regiune statică de comun* : această regiune reprezintă o zonă definită dinamic de către operator sau static, la reconfigurarea sistemului ;

3) *Regiune dinamică* : această regiune este creată dinamic, în momentul execuției unui task, utilizînd directivele de gestiune a memoriei ;

4) *Regiune partajabilă* : această regiune este definită de zona reentrantă (**RO** — cu acces numai în citire), a unui task multiutilizator (în sistemul de operare **MIX-PLUS**).

În figura 5.7 este prezentat schematic spațiul de adresare logic al unui task. Acest spațiu se poate mări sau micșora dinamic în timp, în funcție de numărul de regiuni atașate. Antetul de task și segmentul rădăcină fac parte întotdeuna din regiunea de tip task. Întrucît fiecare regiune se găește într-o zonă contiguă de memorie, aceasta este reprezentată ca un bloc separat.

Task-urile se referă la regiuni prin intermediul unui identificator de regiune. Identificatorul (valoarea) 0 reprezintă întotdeuna regiunea de tip task. Toți ceilalți identificatori reprezintă adresele descriptorilor de atașare (**ATD**) ai unui task la regiuni statice de comun sau dinamice. În sistemul de operare **MIX** un task se poate atașa la maximum 7 regiuni, iar sub **MIX-PLUS** la maximum 23.

Fiecărei regiuni îi corespunde și un bloc de descriere a regiunii (**RDB**), specificat în programul utilizator.

Numărul de regiuni la care se poate ataşa un task pe parcursul execuţiei acestuia este infinit, existînd posibilitatea reproiectării segmentelor virtuale aparţinînd unui task pe diferite regiuni ataşate la un moment dat spaţiului său logic de adresare.

Utilizarea directivelor de gestiune a memoriei permite următoarele acţiuni utilizator asupra regiunilor :

— crearea unei regiuni dinamice, într-o partiţie controlată de sistem **(CRRG$)** ;

— ataşarea task-ului apelant la o regiune statică de comun sau o regiune dinamică, cu nume **(ATRG$)** ;

— detaşarea task-ului apelant de la regiunea specificată cu ştergerea proiecţiilor segmentelor virtuale, corespunzătoare task-ului, din regiune **(DTRG$)** ;

— obţinerea de informaţii privind regiunea specificată **(GREG$)** ;

— trimiterea unei referinţe de ataşare la o regiune (alta decît cea de tip task) către un task specificat **(SREF$)** ;

— recepţionarea unei referinţe de ataşare la o regiune (alta decît cea de tip task), transmisă de către alt task **(RREF$)**.

Aceste directive permit astfel expandarea dinamică a spaţiului de adresare logic ataşat unui task. Cu alte cuvinte, un task poate avea acces la zone logice ce nu fac parte din imaginea sa statică (imaginea executabilă produsă după editarea de legături). Prin proiectarea segmentelor virtuale aparţinînd unui task în zone logice aparţinînd altui task, acestea devin parte integrantă din spaţiul logic de adresare aparţinînd primului task. În mod similar, se pot crea noi regiuni de spaţiu logic în care se pot proiecta segmentele virtuale aparţinînd unui task.

Ataşarea la o regiune reprezintă mijlocul prin care aceasta devine parte integrantă a spaţiului logic de adresare al unui task. Ataşarea are la bază o serie de criterii, bazată pe următorul *mecanism de protecţie şi acces :*

— fiecare regiune are o *mască de protecţie* pentru prevenirea acceselor neautorizate ; masca indică tipurile de acces (citire, scriere, extindere, ştergere) pentru fiecare categorie de utilizatori (sistem, individ, grup, populaţie). Formatul măştii de protecţie este similar cu cel al cuvîntului de protecţie (acces) la un fişier ;

— fiecare regiune are un *proprietar*, identificat prin **UIC**-ul task-ului ce a solicitat crearea regiunii.

Există mai multe modalităţi de *ataşare a unui task la o regiune* :

— *automat*, de regiunile specificate static la editarea de legături ;

— *dinamic*, la o regiune statică de comun sau o regiune dinamică cu nume ;

— prin emiterea unei *referinţe de ataşare* a unui alt task la orice regiune, în spaţiul logic de adresare al task-ului specificat (numai sub **MIX-PLUS**).

Pentru determinarea identificatorului (ID) al unei regiuni ataşate (la task), se forţează o operaţie de ataşare cu specificarea numelui regiunii. Dacă regiunea era ataşată la task, Monitorul setează identificatorul regiunii şi dimensiunea sa (corespunzătoare primei ataşări la task) în blocul de descriere a regiunii **(RDB)**. Metoda este utilă pentru determinarea informaţiilor de descriere ale unui bloc de comun (sau bibliotecă *cluster*) ataşat(ă) task-ului la editarea de legături.

Ataşarea unui task la o regiune previne şi distrugerea acesteia, pînă la momentul detaşării complete, a tuturor task-urilor ce au solicitat acces la regiune.

La detașarea unei regiuni se elimină și toate segmentele virtuale mapate în respectiva regiune.

Distrugerea (eliminarea) unei regiuni dinamice din memorie are loc după ultima operație de detașare a unui task cu acces la regiune.

La crearea unei regiuni dinamice, cu nume, aceasta este colectată în lista blocurilor de comun (CBD). În sistemul de operare **MIX**, alocarea memoriei atașate unei regiuni dinamice se face la creare (de la adresele superioare către cele inferioare ale unei partiții controlate de sistem); regiunile dinamice nu sînt evacuabile.

În sistemul de operare **MIX-PLUS**, memoria necesară unei regiuni dinamice este alocată numai în momentul mapării task-ului la regiune, de către Monitor; regiunile dinamice sînt evacuabile.

Figura 5.8 reprezintă un exemplu posibil de proiecții segmente virtuale — regiuni, corespunzătoare figurilor 5.6 și 5.7.

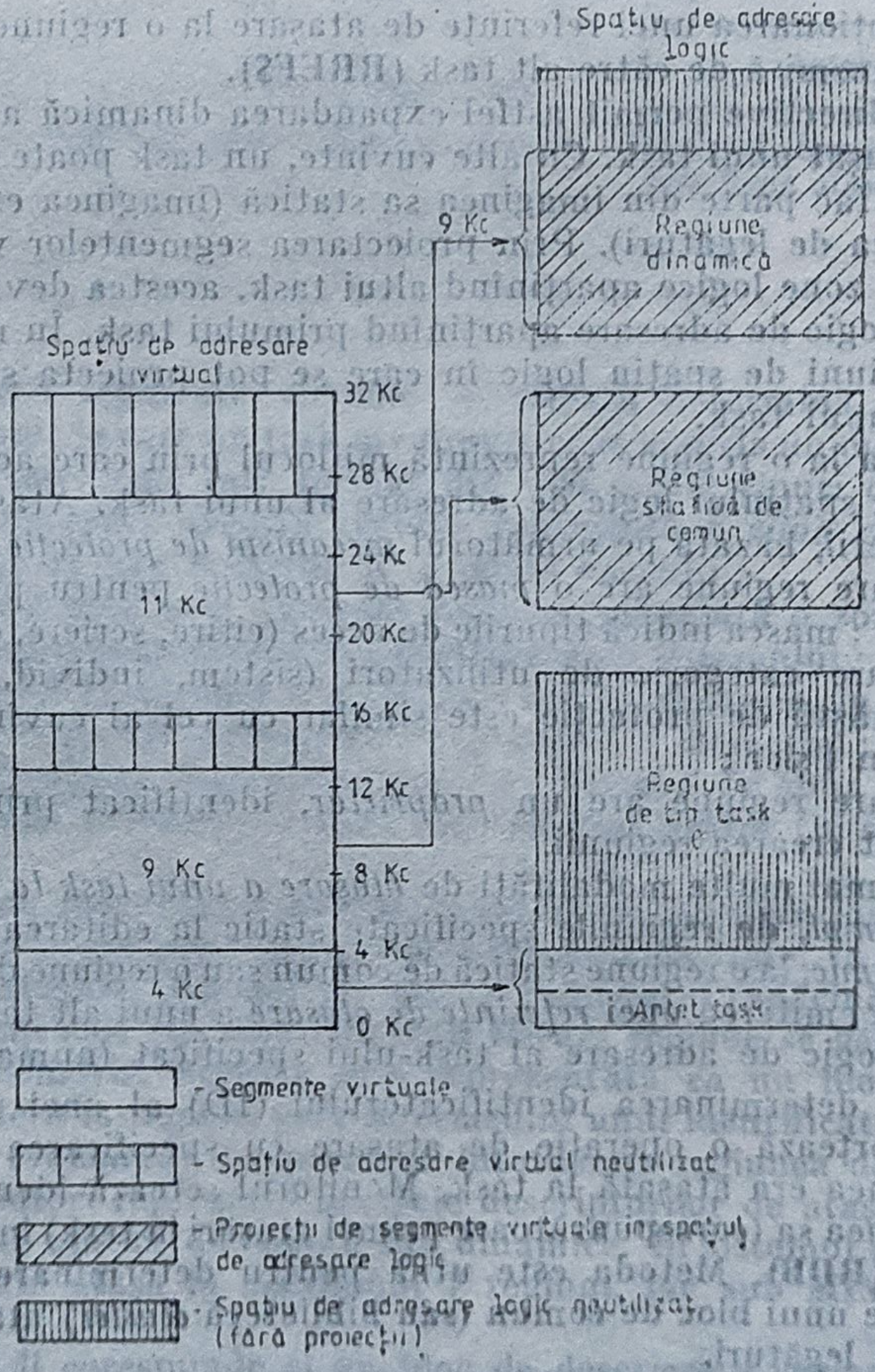


Fig. 5.8, Proiectarea spațiului de adresare virtual în spațiul de adresare logic.

O consecință interesantă a proiectării de segmente virtuale — regiuni o constituie creșterea posibilităților de interacțiune a mai multor task-uri prin intermediul regiunilor partajate.

Un exemplu îl constituie crearea dinamică a unei regiuni de către un task și depunerea de informații în aceasta. Oricare alt task poate avea ulterior acces la aceste informații prin simpla proiectare a unui număr oarecare de segmente virtuale proprii în regiune.

Un alt exemplu îl constituie utilizarea regiunilor statice conținînd rutine (programe) de interes comun. Orice task se poate proiecta, în momentul execuției, pe una sau mai multe rutine necesare, eliminîndu-se necesitatea specificării explicite a acestora la editarea de legături.

5.8. Maparea task-urilor privilegiate

Întrucît task-urile privilegiate, în sistemul de operare **MIX**, au drepturi speciale de acces la memorie (la Nucleul sistemului și la pagina de intrare/ieșire), alocarea spațiului virtual atașat acestora este diferită de cea a task-urilor utilizator obișnuite.

Precizarea statutului privilegiat al unui task se face la editarea de legături, prin folosirea comutatorului **/PR**, aplicat pe fișierul imagine task (**TSK**).

Alocarea spațiului virtual este diferită, funcție de argumentul opțional ce însoțește comutatorul **/PR : 0**, **4** sau **5** (valoare implicită), reprezentînd numărul registrului de relocare (**APR**) utilizat drept bază în alocarea spațiului virtual.

La precizarea argumentului 0 (**/PR : 0**), task-ul este marcat ca privilegiat, dar nu este mapat la Monitor și pagina de intrare/ieșire. Memoria virtuală atașată acestuia începe la locația zero și se poate extinde pînă la valoarea de 64 kocteți — 64 octeți (1777778).

Un asemenea task privilegiat se poate executa numai în starea utilizator și nu se poate comuta pe starea sistem (Kernel). Scopul mapării este acela de a preveni coruperea Monitului și a structurilor de date asociate, în cazul în care se dorește o serie de drepturi speciale de acces :

— ocolirea protecției de acces la fișiere ;

— utilizarea directivei Monitor de modificare a priorității unui task (**ALTP$**) ;

— utilizarea directivei Monitor **GIN$ (MIX-PLUS)**, de obținerea a unor informații atașate structurilor de date sistem ;

— utilizarea directivei Monitor **SWST$**, de trecere din starea curentă (utilizator) în starea sistem (Kernel) ;

— execuția unei directive ce precizează un task destinatar diferit de cel curent ;

— specificarea unui nume de dispozitiv în directivele **SPWN$, RPOI$** ;

— efectuarea unor operații de intrare/ieșire logice asupra unor volume, indiferent de starea de montare sau alocare a acestora.

Exemple de asemenea task-uri în sistemele de operare **MIX/MIX-PLUS** sînt utilitarele : **BRU, FMT, BAD, ICM (MIX-PLUS)**, etc.

La precizarea argumentului 4 sau 5 (**/PR : 4** sau **/PR : 5**), task-ul este marcat ca privilegiat și se mapează la Monitor și la pagina de intrare/ieșire. În cadrul spațiului virtual atașat task-ului privilegiat, Editorul de legături rezervă **APR**-ul 7 pentru maparea paginii externe și **APR**-urile 1 la 3 (sau 4) pentru maparea Monitorului.

Alegerea **APR**-ului 4 sau 5 ca bază pentru adresele virtuale din cadrul task-ului este dependentă de dimensiunea zonei rezidente a Monitorului. Dacă Monitorul este mai mic sau egal cu 16 Kcuvinte, (ocupînd **APR**-urile 0 la 3), se precizează argumentul 4 ; dacă Monitorul este cuprins între 16 și 20 Kcuvinte (ocupînd **APR**-urile 0 la 4), se precizează argumentul 5. Monitorul sistemului de operare **MIX-RT** poate avea o dimensiune mai mică sau egală cu 16 sau 20 Kcuvinte ; Monitoarele sistemelor de operare **MIX/MIX-PLUS** au întotdeuna o dimensiune egală cu 20 Kcuvinte.

În funcție de argumentul specificat, Editorul de legături aplică o bază egală cu 100 000 (16 Hc, **/PR : 4**) sau 120 000 (20 Kc, **/PR : 5**) tuturor adreselor virtuale din cadrul task-ului. De asemenea, spațiul virtual propriu task-ului poate avea dimensiunea de 12 Kcuvinte (**/PR : 4**) sau 8 Kcuvinte (**/PR : 5**), tipică pentru sistemele de operare **MIX/MIX-PLUS.** Începutul task-ului marchează sfîrșitul zonei Monitor, iar sfîrșitul task-ului începutul paginii de intrare/ieșire.

Dacă task-ul privilegiat urmează a accesa pagina de intrare/ieșire, dimensiunea acestuia nu poate depăși 12 Kc (sau 8 Kc). Dacă dimensiunea task-ului depășește această valoare, Editorul de legături forțează alocarea **APR**-ului 7 în corpul task-ului și nu peste pagina de intrare/ieșire. În acest caz, la instalarea task-ului se emite un mesaj de atenționare specific (ce poate fi inhibat, la editarea de legături, prin folosirea comutatorului **/-IP**). Depășirea dimensiunii de 16 Kc (**/PR : 4**) sau 12 Kc (**/PR : S**), duce la eroare la editarea de legături. Exemple de asemenea task-uri în sistemele de operare **MIX/MIX-PLUS** sînt componentele sistem : **INI, INS, MOU**, etc.

Un task privilegiat, mapat cu argumentul 4 sau 5, are următoarele *drepturi speciale de acces* :

— poate apela rutinele Monitor ;
— poate accesa și modifica structurile de date Monitor ;
— poate accesa pagina de intrare/ieșire ;
— poate efectua operații de intrare/ieșire logice la volumele montate ;
— poate solicita trecerea din starea curentă (utilizator), în starea sistem (Kernel) prin execuția macroinstrucțiunii **$SWSTK** ; la trecerea în starea sistem, registrele generale ale task-ului sînt salvate, putînd fi modificate de utilizator. Refacerea acestora are loc automat, la ieșirea din starea sistem **(RTS PC)**. Secvența de instrucțiuni executată în starea sistem poate accesa și modifica în mod unic, protejat, structurile de date Monitor. Acest mod de lucru permite evitarea **interblocărilor (deadlock)** sistem prin serializarea accesului la structurile unice de date Monitor și este folosit de toate task-urile privilegiate ale familiei de sisteme de operare **MIX**.

O situație specială este reprezentată de task-urile privilegiate (mapate cu argumentele 4 sau 5), ce utilizează directive de gestiune a memoriei **(PLAS)**. Utilizînd aceste directive, un task privilegiat poate remapa, pe diferite regiuni, orice număr din **APR**-urile dedicate accesului la Monitor sau pagina de intrare/ieșire. În acest caz, o remapare eronată poate duce la erori sistem, greu de detectat și eliminat. În general, la demaparea unui segment virtual, anterior

mapat la Monitor sau pagina de intrare/ieşire, Monitorul restaurează maparea iniţială. Este interzisă, de aceea, demaparea **APR0** de către task-urile privilegiate, întrucît această zonă conţine locaţia **$DSW**, structurile de descriere a segmentelor, zonele de reentranţă task, etc.

Task-urile privilegiate banale (**/PR : 0**), ce nu au facilităţi de mapare la Monitor, îşi pot extinde accesul la Monitor şi structurile sale de date prin utilizarea directivei Monitor **SWST$**.

Ca şi în cazul directivei **CINT$**, se poate specifica în acest caz, ca argument de apel **SWST$**, adresa unei rutine utilizator ce urmează a fi mapată la Monitor. În urma execuţiei directivei, rutina specificată va fi mapată cu **APR5**, modul *Kernel*, şi poate utiliza gama de adrese virtuale 120000 la 137777 (lungime 4 kcuvinte, inclusiv date locale accesate de către rutină). Întrucît execuţia rutinei utilizator reprezintă o continuare a execuţiei directivei **SWST$**, ieşirea din rutină (şi implicit refacerea mapării iniţiale) se face, de către Monitor, prin programarea instrucţiunii **RTS PC**.

5.9. Maparea bibliotecilor „cluster"

Termenul de *biblioteci „cluster"* descrie o funcţie şi o structură creată de Editorul de legături, ce permite unui task maparea dinamică a unor regiuni partajate, la momentul execuţiei.

Funcţia de *„cluster"* permite unui task să acceseze (utilizeze) mai multe biblioteci de rutine partajate (cum ar fi, de exemplu biblioteca modulelor de acces **FCS(FCSANS) RMS(RMSRES)**, biblioteca **FMS (FMSRES)**, rutinele de execuţie **F77 (F77RES)**, etc.) mapate cu acelaşi segment virtual (încep la aceeaşi adresă în spaţiul virtual al task-ului).

Deşi la un moment dat, numai una din bibliotecile *„cluster"* este mapată în spaţiul de adresare al task-ului, maparea unei alte biblioteci se face automat, în momentul referirii acesteia.

În proiectarea şi construirea unor biblioteci *„cluster"* partajabile trebuiesc respectate o serie de reguli :

— la editarea de legături a task-ului cu opţiunea *„cluster"*, toate bibliotecile, cu excepţia primei, trebuiesc incluse într-o structură de segmente rezidente în memorie ; dacă bibliotecile, inclusiv prima, sînt segmentate, se creează o structură de segmente prin specificarea unei rădăcini nule ;

— programul utilizator trebuie să rezolve intern (prin vectorizare indirectă), referinţele între bibliotecile aparţinînd configuraţiei *„cluster"* ;

— simbolurile revectorizate, trebuie eliminate din fişierul de simboluri **(STB)** ataşat unei biblioteci ;

— procedura de apel a unei biblioteci nu trebuie să paseze parametri în stivă ;

— toate bibliotecile trebuie să fie de tip **PIC** sau să fie construite cu aceiaşi adresă de bază (absolute) ;

— nu se permit tratări de întreruperi asincrone **(AST)** în bibliotecile *„cluster"*.

Pentru utilizarea mai multor biblioteci în regim *„cluster"*, este necesară construirea separată a acestora, plasarea fişierelor **TSK** şi **STB** în catalogul

LB : [1,1] și construirea task-ului cu specificarea opțiunii **CLSTR**. La editarea de legături a task-ului, pentru fiecare bibliotecă cu structură rezidentă în memorie, se construiesc în rădăcina task-ului : vectori de autoîncărcare, descriptori de segmente virtuale și un descriptor de regiune.

Utilizarea bibliotecilor „*cluster*" minimizează accesul la spațiul virtual și necesită, de cele mai multe ori, numai un segment virtual.

Numărul maxim de hihlioteci într-un „*cluster*" este limitat la 6 și numărul minim la 2.

5.10. Facilități de mapare rapidă în sistemul de operare MIX-PLUS

Sistemul de operare **MIX-PLUS** extinde facilitățile directivelor de gestiune a memoriei cu opțiunea de **mapare rapidă**, prin includerea unor algoritmi specifici în corpul directivei Monitor **MAP$**. Performanțele de viteză obținute prin utilizarea acestui mecanism sînt semnificative : de la 10 la 30 de ori mai repede decît în cazul utilizării directivei **MAP$**.

În acest caz, interfața de acces utilizator este instrucțiunea **IOT** (și nu directiva **MAP$**), iar argumentele de apel se transmit în registrele generale (și nu în **DPB**), economisindu-se 200—300 instrucțiuni executate în directiva **MAP$**. Utilizarea interfeței **IOT** exclude, pentru task-ul utilizator, alte funcții sau modalități de comunicare implementabile cu ajutorul **IOT**.

Pentru utilizarea facilității de mapare rapidă, antetul task-ului *(header)* se extinde, la editarea de legături (**/FM**), cu o *zonă de mapare rapidă*. Mărirea dimensiunii antetului de task (semnificativă deseori) face necesară declararea acestuia în partiție — *antet extern* —, cu ajutorul opțiunii **/XH** a editorului de legături.

Înaintea utilizării facilității de mapare rapidă, task-ul trebuie să creeze (**CRAW$**) și mapeze (**MAP$**) primar segmentul virtual ce se dorește a fi accesat, Accesul rapid la diferitele zone ale segmentului virtual (ce poate fi mapat în spațiul I sau **D** *Utilizator*) se poate face pe o lungime variabilă (multiplu de *64 octeți*), specificată în argumentele de apel.

Limbajele de nivel înalt au acces la ficilitățile de mapare rapidă prin intermediul rutinelor de bibliotecă **FMAP** și **FMAPL**.

Pentru un task utilizator obișnuit, caracteristicile de mapare rapidă pot fi stabilite și dinamic, la instalare, cu opțiunea **/FMAP = YES**.

5.11. Task-uri privilegiate vectorizate

În scopul asigurării independenței unui task privilegiat față de variante de sistem existentă, începînd cu versiunea **MIX-PLUS V2.0** este disponibilă și utilizată pe larg facilitatea de *vectorizare a task-urilor privilegiate și a driverelor* (sistem și utilizator).

Vectorizarea componentelor privilegiate (mapate la Monitor) permite o integrare ușoară a acestora în diferite variante (sau versiuni) ale sistemului

de operare **MIX-PLUS**, fără a fi necesare operații suplimentare (de asamblare sau editare de legături). Astfel, un task privilegiat vectorizat este unic, indiferent de tipul de sistem folosit : **MIX-PLUS** cu spații I și D (pentru calculatoarele **I-102/4M, I-106, I-1016**), **MIX-PLUS** fără spații I și D (**CORAL 4021, DP15, DP21**) sau **MIX-RT** (**I-1016**). Același task va putea fi preluat în versiunile ulterioare ale sistemului de operare fără modificări.

Realizarea independenței unui task privilegiat față de Monitor este posibilă prin vectorizarea referințelor la acesta (nume de rutine și deplasări globale la tabelele sistem). Vectorizarea impune transformarea tuturor referințelor la Monitor în *deplasări (offset-uri)* relative la un *vector sistem unic*, ce conține adresele reale ale referințelor la Monitor.

Deplasările relative la vectorul sistem sînt imuabile (unice și fixe) și sînt conținute în fișierul **MIXVEC.STB**. Vectorul sistem ale cărui intrări sînt adresele reale în Monitor este conținut într-un comun Monitor special (numit **VECMIX**), încărcat la configurarea sistemului (cu ajutorul **VMR**).

Editarea de legături a unui task privilegiat se va face cu fișierul **MIXVEC.STB** și nu **MIXPLS.STB** (ce conține deplasări în vectorul sistem și nu adrese reale în Monitor). De remarcat faptul că un asemenea task nu poate fi utilizat pe un sistem ce nu suportă facilitatea de vectorizare (cum ar fi **MIX V2.0, MIX-PLUS V1.0**). În acest caz, task-ul (nemodificat) trebuie editat cu fișierul de simboluri **MIX(PLS).STB**. În practică, sînt distribuite în mod standard ambele fișiere de simboluri **MIXPLS.STB** și **MIXVEC.STB**, recomandîndu-se utilizarea facilității de vectorizare (folosită pe larg în sistem atît pentru realizarea extensiilor încărcabile ale Monitorului: **MDS, SEC, CDAxy, PAR, POWER, SHADOW**, cît mai ales pentru realizarea driverelor și task-urilor privilegiate : **RMD, MTAACP, COT** etc.).

Pentru realizarea vectorizării este necesară modificarea codului task-urilor privilegiate în felul următor :

a) Construirea, în task, a unui *vector local* constituit din referințele dorite la Monitor. Fiecare intrare în vector este reprezentată de un cuvînt (**.WORD**) ; referirea intrării se face printr-o etichetă locală aleasă sugestiv, prin omiterea, de exemplu, a semnului dolar ($) din referința la Monitor :

CRHDR : .WORD $CRHDR

Vectorul local este prefațat de un cuvînt nul (**.WORD 0**), ce indică situația inițială (nu a fost realizată inițializarea vectorului de intrări). De remarcat că referințele globale la Monitor (rezolvate din fișierul **MIXVEC.STB**) sînt de fapt deplasări în vectorul sistem (conținut în comunul **VECMIX**).

b) Schimbarea, în cadrul task-ului, a tuturor referințelor directe la Monitor cu referințe indirecte prin intermediul intrărilor din vectorul local :

MOV @ CRHDR, R0, în loc de :
MOV $CRHDR, R0

c) Construirea, la începutul task-ului, a unei secvențe conținînd directiva Monitor **GINT**, care realizează translatarea deplasărilor din intrările vectorului local (atașat task-ului) în adrese reale din cadrul Monitorului. Execuția direc-

tivei **GIN$** duce la înlocuirea fiecărei deplasări cu adresa reală din Monitor. Translatarea vectorului este marcată prin înscrierea unei valori diferite de zero în primul cuvînt al vectorului (aceasta permite ca, la următoarele lansări în execuţie ale task-ului, să nu se mai efectueze noi translatări).

5.12. Structurile de task-uri şi gestiunea memoriei

În urma unei editări de legături (**TKB**) se obţine o imagine task ce poate avea una din următoarele structuri :

— contiguă ;
— segmentată, cu segmente rezidente pe disc ;
— segmentată, cu segmente rezidente în memorie ;
— multi-utilizator (pe sistemul **MIX-PLUS**).

5.12.1. Structura contiguă

Această structură implică aducerea în memorie a tuturor elementelor componente ale unui task.

Structura este avantajoasă pentru sistemele de operare cu memorii mari, în care pot încăpea task-uri de mari dimensiuni, sau în care cerinţele exploatării în timp real presupune rezidenţa completă a programului în memorie.

În scopul obţinerii unor timpi de răspuns mai mici, o bună parte din componentele privilegiate ale sistemului de operare **MIX** au structuri contigue şi se leagă la o bibliotecă partajată de module de acces **FCS (FCSANS)**.

O astfel de structură contiguă nu poate fi însă utilizată dacă depăşeşte spaţiul fizic de 28 kcuvinte (sisteme fără relocare) sau spaţiiul virtual de 32 kcuvinte (sisteme cu relocare). În acest caz, există următoarele soluţii de structură :

— fragmentarea task-ului în mai multe bucăţi : un task principal legat la mai multe zone de comun sau biblioteci partajate, sau o structură complexă de task-uri cooperante.

— structuri de segmente.

Pentru exemplificarea diferitelor structuri de task-uri în fig. 5.9 se imaginează o schemă a legăturilor logice existente în cadrul unui task.

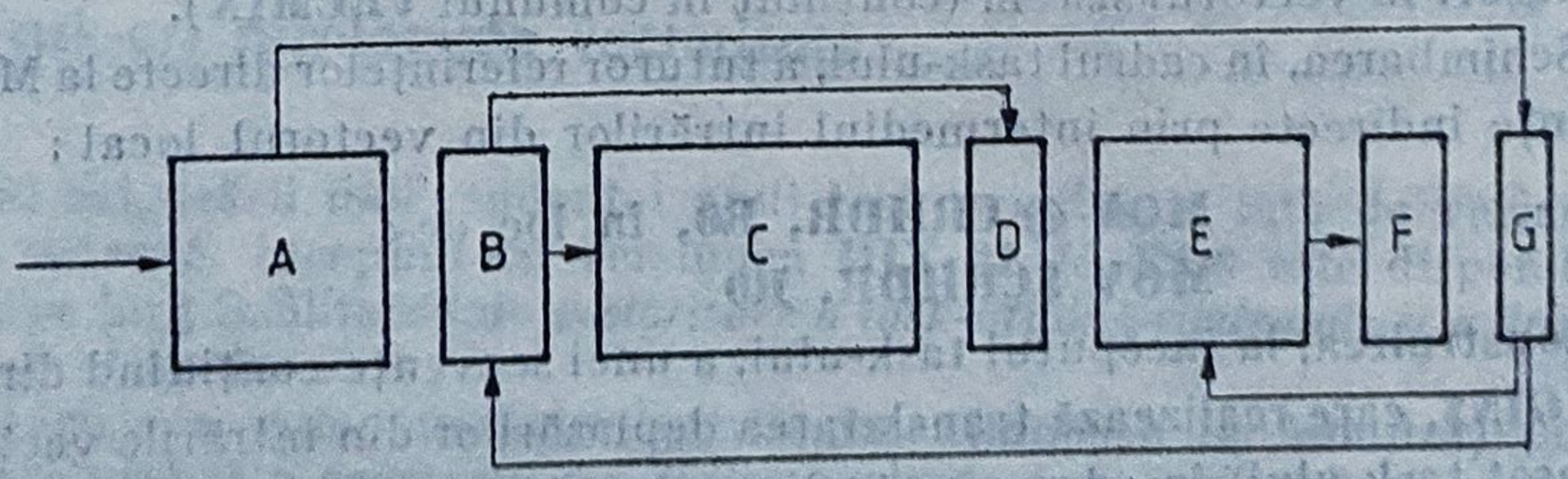


Fig. 5.9. Schema legăturii logice între modulele unui task

Fig. 5.10 prezintă structura contiguă a unui task.

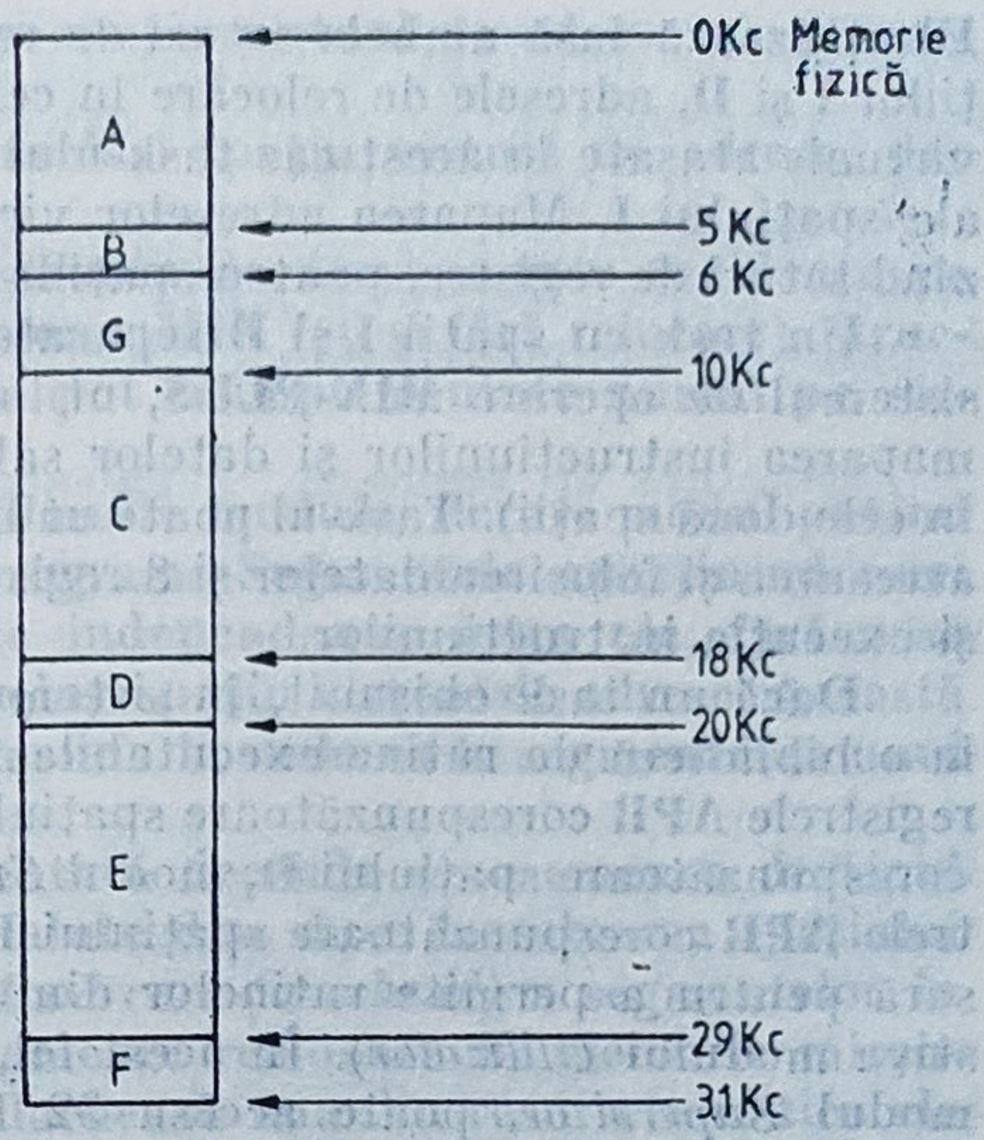


Fig. 5.10. Structura contiguă a unui task

Sistemul de operare **MIX-PLUS** exploatează în mod eficient performanțele noilor Unități de relocare și protecție a memoriei existente pe ultimele minicalculatoare românești, permițînd dublarea sau triplarea spațiului de adresare al unui task fără utilizarea structurilor segmentate, rezidente disc sau în memorie.

Minicalculatoarele **I-106** și **I-1016** permit maparea unor biblioteci de rutine utilizator sau sistem (cum ar fi serviciile de acces la fișiere, **FCS** sau **RMS**), cu *acces numai în citire*, utilizînd registrele modului *Supervizor*. Aceasta permite extinderea spațiului normal de adresare au unui task (de 32 Kcuvinte) cu încă 32 Kcuvinte de cod cu acces numai în citire (spațiul **I** al modului *Supervizor*). Performanțe mai mari se obțin prin segmentarea bibliotecilor de rutine mapate în modul *Supervizor* decît prin segmentarea task-ului utilizator.

Sistemul de operare **MIX-PLUS** permite de asemenea separarea zonelor de instrucțiuni (**I**) și date (**D**) aparținînd unui task, ceea ce extinde posibilitățile de adresare ale acestuia la 64 Kcuvinte (32 Kcuvinte de instrucțiuni și 32 Kcuvinte de date). Utilizînd și bibliotecile mapate în modul *Supervizor*, un task separat pe spații **I** și **D** poate adresa maxim 96 Kcuvinte de memorie — ceea ce duce la creșterea performanțelor sistemului și simplifică dezvoltarea de programe.

În primul caz, numărul de segmente virtuale atașat unui task poate crește pînă la 16 (8 segmente pentru maparea spațiului de instrucțiuni și 8 segmente pentru maparea spațiului de date), într-un spațiu virtual de 64 Kcuvinte. Un asemenea task are cel puțin două segmente virtuale (cu numerele 0 și 1), corespunzătoare mapării antetului și segmentului rădăcină în cele două spații (**I** și **D**).

În cel de-al doilea cauz, numărul de segmente virtuale atașat unui task poate crește pînă la 24 (8 segmente pentru maparea spațiului de instrucțiuni în modul *Utilizator*, 8 segmente pentru maparea spațiului de date în modul *Utilizator*, 8 segmente pentru maparea spațiului de instrucțiuni în modul *Supervizor*), într-un spațiu virtual de 96 Kcuvinte. Numărul de segmente virtuale rezervate (0 și 1) nu crește însă, maparea spațiului de instrucțiuni în modul *Supervizor* făcîndu-se dinamic, la apelarea unei referințe din această zonă.

Un task obișnuit, ce se execută în modul *Utilizator*., pe un sistem **MIX/MIX-PLUS** fără spații **I** și **D** separate, folosește numai registrele spațiului **I** pentru maparea instrucțiunilor și datelor sale. Un asemenea task se poate executa pe un sistem **MIX-PLUS**, cu spații **I** și **D** separate, fără modificări.

El utilizează însă ambele seturi de registre de relocare pentru maparea spațiilor **I** și **D**, adresele de relocare în cele două spații fiind identice. Segmentele virtuale atașate în acest caz task-ului se referă numai la registrele de relocare ale spațiului **I**. Maparea adreselor virtuale în adrese logice se face însă utilizînd seturi de registre, pentru spațiile **I** și **D**, conținutul acestora fiind identic.

Un task cu spații **I** și **D** separate, ce se execută în modul *Utilizator* sub sistemul de operare **MIX-PLUS**, utilizează seturi de registre separate pentru maparea instrucțiunilor și datelor sale (adresele de relocare nefiind identice în cele două spații). Task-ul poate utiliza 8 registre **APR** ale spațiului **D** pentru accesarea și folosirea datelor și 8 registre **APR** ale spațiului **I** pentru accesarea și execuția instrucțiunilor.

Dacă un task obișnuit, în sistemul de operare **MIX-PLUS**, se conectează la o bibliotecă de rutine executabile în modul *Supervizor*, Monitorul copiază registrele **APR** corespunzătoare spațiului **I**, modul *Utilizator*, în registrele **APR** corespunzătoare spațiului **D**, modul *Supervizor* și mapează biblioteca, cu registrele **APR** corespunzătoare spațiului **I**, modul *Supervizor* (maparea este necesară pentru a permite rutinelor din modul *Supervizor* să acceseze datele și stiva modului *Utilizator*). În acest fel, un program ce execută instrucțiuni în modul *Supervizor*, poate accesa 32 Kcuvinte din spațiul său propriu prin intermediul registrelor **APR** ale spațiului **D** și 32 Kcuvinte din spațiul bibliotecii de rutine prin intermediul registrelor **APR** ale spațiului **I**.

Dacă un task cu spații **I** și **D** separate, în sistemul de operare **MIX-PLUS**, se conectează la o bibliotecă de rutine executabile în modul *Supervizor*, Monitorul copiază registrele **APR** corespunzătoare spațiului **D**, modul *Utilizator*, în registrele **APR** corespunzătoare spațiului **D**, modul *Supervizor*. Rutinele din modul *Supervizor* pot accesa spațiul de date *Utilizator* și spațiul de instrucțiuni *Supervizor* prin intermediul registrelor **APR** corespunzătoare spațiului *I*, modul *Supervizor*.

În rezumat, se poate spune că încărcarea registrelor de mapare din spațiile **I** și **D** *Supervizor* este realizată de Monitor în felul următor :

— spațiul **I** *Supervizor* mapează întotdeuna rutinele bibliotecii *Supervizor* ;

— spațiul **D** *Supervizor* mapează spațiul **D** *Utilizator* al programului, iar în cazul în care acesta nu are spații **I** și **D** separat, mapează spațiul **I** *Utilizator*.

Pe sistemele cu spații **I** și **D** separate, task-urile ce nu utilizează spațiul **D** (**/-ID**) se execută de obicei cu cele două spații **I** și **D** suprapuse. Cu toate acestea, task-urile pot crea segmente virtuale mapate în spațiul **D**, ceea ce duce la creșterea spațiului virtual total fără utilizarea facilităților integrale oferite de spațiile **I** și **D**. Regiunile de comun statice la care poate avea acces un task sînt de obicei mapate numai în spațiul **D**. Bibliotecile de rutine partajate sînt de obicei mapate atît în spațiul **I** cît și în spațiul **D**.

5.12.2. Structuri segmentate

În acest caz, un task este împărțit în mai multe segmente :

— un segment principal (numit rădăcină — **ROOT**), rezident tot timpul în memorie ;

— un număr variabil de segmente secundare :

a) *rezidente pe disc*, partajînd același spațiu virtual de adrese și aceeași memorie fizică între ele, sau

b) *rezidente în memorie*, partajînd același spațiu virtual de adrese, dar nu și aceeași memorie fizică.

De remarcat că spre deosebire de structurile de segmente rezidente pe disc, ce pot exista pe toate tipurile de sisteme **MIX** (cu și fără relocare), structurile de segmente rezidente în memorie pot fi implementate numai pe sistemele cu relocare.

Segmentele constau din una sau mai multe module, fiecare modul putînd avea una sau mai multe secțiuni de program. Segmentele aparținînd unei structuri segmentate trebuie să fie logic independente între ele, fără a se referi unul pe altul întrucît partajează același spațiu virtual de adrese. Această independență logică trebuie avută întotdeauna în vederea la segmentarea unui program de dimensiuni mari.

Alegerea tipului de structură segmentată depinde de asemenea de tipul aplicației. Alegerea unei structuri segmentate disc duce la salvarea spațiului fizic, dar impune *timpi suplimentari de încărcare (overhead)* a segmentelor în memorie. Alegerea unei structuri cu segmente rezidente în memorie micșorează timpii de mapare/remapare, dar blochează una din resursele critice ale sistemului — memoria fizică.

Precizarea structurii de segmente se face la editarea de legături (**TKB**), cu ajutorul unor directive specifice **TKB**, colectate într-un fișier asociat, tip **ODL**.

Fig. 5.11 prezintă structura segmentată a unui task.

Structură de segmente, rezidente pe disc

Structurile rezidente pe disc conservă spațiul de adresare virtual și memoria fizică prin partajarea acestora între segmente.

Deși această soluție prezintă avantajul unei economii importante de memorie, ea necesită însă un timp sistem îndelungat pentru încărcarea acestor

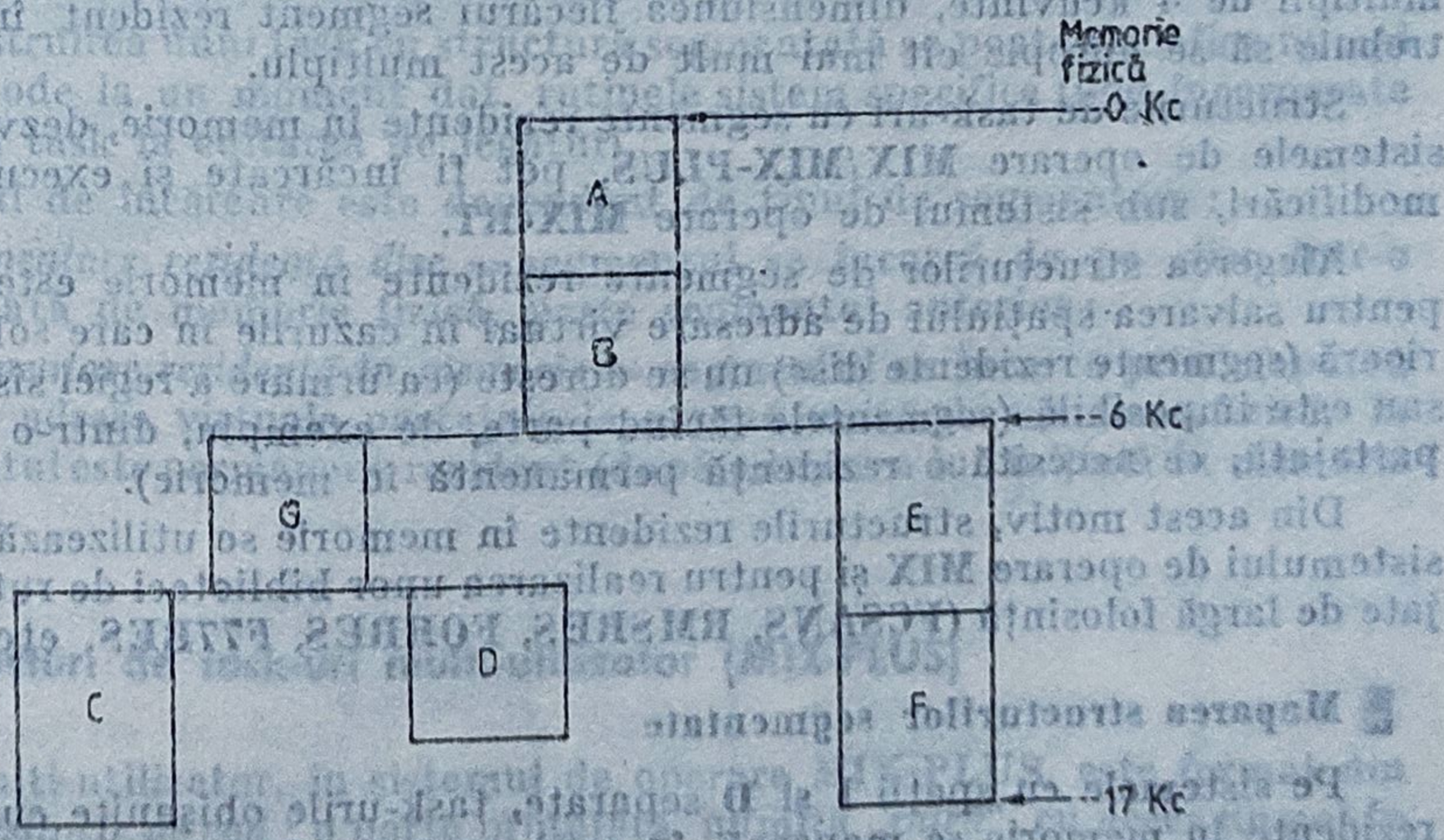


Fig. 5.11. Structura segmentată a unui task

segmente în memorie în momentul referirii lor (din rădăcină). Mai mult, o asemenea structură nu poate fi deloc exploatată în cazul sistemelor **MIX-RT** fără disc în configurație.

Această structură este însă reținută pentru sistemele **MIX/MIX-PLUS**, în care majoritatea utilitarelor, compilatoarelor și subsistemelor conversaționale sînt implementate cu structuri de segmente, rezidente pe disc.

Structură de segmente, rezidente în memorie

Segmentele rezidente în memorie partajează același spațiu de adresare virtual (ca și segmentele rezidente disc), dar nu și aceeași memorie fizică.

Segmentele structurii sînt încărcate de pe disc în memorie la prima referire a unui segment din rădăcină. Segmentele rămîn în continuare în memorie și referințele la ele se rezolvă prin remapare. În cazul sistemului de operare **MIX-RT**, încărcarea segmentelor rezidente în memorie are loc la instalarea task-ului cu această structură.

O astfel de structură poate fi implementată numai pe sistemele cu relocare. Fiecare segment rezidă într-o zonă separată de memorie fizică, aliniată la un multiplu de 32 (sau 256) cuvinte. În momentul referirii la un segment, prin utilizarea unor directive de gestiune a memoriei în rutinele de încărcare, aceasta se reduce la schimbarea proiectării adreselor virtuale, corespunzătoare segmentului, în memoria fizică permanentă atașată acestuia.

Corespondența între spațiile virtual și logic atașate task-ului utilizator se face prin proiectarea automată a diferitelor segmente virtuale atașate unui task pe diferite zone ale spațiului logic reprezentate de segmentele rezidente în memorie.

În acest context, rădăcina și celelalte segmente rezidente în memorie reprezintă regiunea de tip task a spațiului de adresare logic, proiectată pe memoria fizică existentă.

Utilizarea neatentă a unor astfel de structuri duce la o ineficientă alocare a memoriei virtuale. Întrucît memoria virtuală este alocată în blocuri, pe multipli de 4 kcuvinte, dimensiunea fiecărui segment rezident în memorie trebuie să se apropie cît mai mult de acest multiplu.

Structurile de task-uri cu segmente rezidente în memorie, dezvoltate sub sistemele de operare **MIX/MIX-PLUS**, pot fi încărcate și executate, fără modificări, sub sistemul de operare **MIX-RT**.

Alegerea structurilor de segmente rezidente în memorie este necesară pentru salvarea spațiului de adresare virtual în cazurile în care soluția anterioară (segmente rezidente disc) nu se dorește (ca urmare a regiei sistem mari) sau este imposibilă (segmentele făcînd parte, de exemplu, dintr-o bibliotecă partajată, ce necesită o rezidență permanentă în memorie).

Din acest motiv, structurile rezidente în memorie se utilizează în cadrul sistemului de operare **MIX** și pentru realizarea unor biblioteci de rutine partajate de largă folosință (**FCSANS, RMSRES, FORRES, F77RES,** etc.).

Maparea structurilor segmentate

Pe sistemele cu spații **I** și **D** separate, task-urile obișnuite cu segmente rezidente în memorie se mapează, în mod normal, identic în spațiile **I** și **D** (suprapuse fizic). Pentru maparea unui task cu segmente rezidente în memorie,

cu spaţii **I** şi **D** separate, Editorul de legături alocă minimum trei descriptori de segmente (ferestre de adrese) virtuale, iniţializate astfel :

— **WDB 0** = rădăcina task-ului, spaţiul **I** ;
— **WDB 1** = antetul task-ului, stiva şi rădăcina task-ului, spaţiul **D** ;
— **WDE 2** = segment rezident în memorie ;
— **WDB 3** = segment rezident în memorie ;

. . .

Segmentele rezidente în memorie nu sînt separate şi se mapează atît în spaţiul **I** cît şi în spaţiul **D**.

Structură segmentată combinată

Pentru o serie de aplicaţii în care se urmăreşte atît conservarea spaţiului virtual cît şi a memoriei fizice, la timpi de răspuns medii, se poate alege o structură segmentată combinată, cu segmente rezidente disc şi rezidente în memorie.

În aceste cazuri, se recomandă, pe cît posibil, separarea celor două tipuri de structuri pe co-arbori separaţi, prin precizări specifice la editarea de legături.

Metode de încărcare a segmentelor

În sistemul de operare **MIX** se utilizează două metode distincte pentru încărcarea segmentelor rezidente disc şi memorie :

— *încărcarea automată* = efectuată de rutine sistem, apelate automat în momentul specificării unor referinţe la segmente (metodă unică pentru sistemul de operare **MIX-RT** cu segmente rezidente în memorie) ; tratarea şi recuperarea erorilor de încărcare se face automat ;

— *încărcarea manuală* = efectuată prin apeluri explicite, existente în task-ul utilizator, către rutinele sistem de încărcare a segmentelor ; tratarea şi recuperarea erorilor de încărcare rămîne în sarcina utilizatorului.

La construirea unui task cu structură segmentată se poate specifica numai una din metode la un moment dat, rutinele sistem specifice fiind încorporate în imaginea task la editarea de legături.

Procesul de încărcare este dependent de tipul de segmentare :

— *segmentare rezidentă disc* = segmentul se încarcă de pe disc într-o zonă partajată de memorie fizică, peste segmentul anterior ;

— *segmentare rezidentă în memorie* = segmentul se încarcă prin maparea unui set de adrese virtuale partajate la o zonă unică de memorie fizică, în care segmentul este permanent rezident (după aducerea lui iniţială în memorie de pe disc).

5.12.3. Structuri de task-uri multi-utilizator (MIX-PLUS)

Un task multi-utilizator, în sistemul de operare **MIX-PLUS**, este format din *două zone distincte de cod :* o parte protejată, numită *„pură"*, cu acces numai în citire (**RO** = read-only), şi o parte, numită *„impură"*, cu acces în citire/scriere

(**RW** = read/write). La lansarea inițială în execuție a unui task multi-utilizator, este adusă în memorie cîte o copie a fiecărei zone de cod (**RO** și **RW**). Cererile suplimentare de lansare în execuție duc numai la duplicarea porțiunii „impure" și la partajarea porțiunii „pure", rezultînd o utilizare mai eficientă a resursei memorie internă.

În urma editării de legături a unui task multi-utilizator (/**MU**), acesta este împărțit în două regiuni :

— regiunea 0 = conține partea „impură" (**RW**) ;

— regiunea 1 = conține partea „pură" (**RO**).

Ca și în cazul celorlalte structuri de task-uri, editorul de legături utilizează atributele de acces ale secțiunilor de program (**.PSECT**) pentru plasarea acestora în imaginea multi-utilizator. Spațiul de adresare virtual este separat în porțiuni **RO** și **RW**. Spre deosebire de un task obișnuit, partea pură, cu acces numai în citire, este protejată hardware. Porțiunea „impură" este separată de porțiunea „pură", în regiuni separate, plasate la cele două extremități ale spațiului de adresare virtual al task-ului.

Structurile de task-uri multiutilizator pot fi, de asemenea :

— *contigue*

— *segmentate*

În cazul unei structuri contigue sau segmentate, cu segmente rezidente disc, partea „*pură*" este formată din secțiunile de program **RO** ale rădăcinii, iar partea „*impură*" din secțiunile de program **RW** ale rădăcinii (inclusiv antetul de task și stiva) și toate celelalte segmente ale task-ului.

În cazul unei structuri segmentate, cu segmente rezidente în memorie, porțiunea „pură" (**RO**) a imaginii task poate fi alcătuită din :

— secțiunile de program „pure" ale segmentului rădăcină ;

— ramurile structurii segmentate, dacă acestea sînt complet rezidente în memorie și conțin numai secțiuni de program „pure".

— structuri de co-arbore, dacă acestea sînt în întregime rezidente în memorie și formate numai din secțiuni de program „pure".

Toate celelalte segmente ale structurii, secțiunile de program „impure" (**RW**) ale rădăcinii, antetul de task și stiva formează zona „impură" a task-ului multi-utilizator.

Figura 5.12 prezintă structura unui task multi-utilizator.

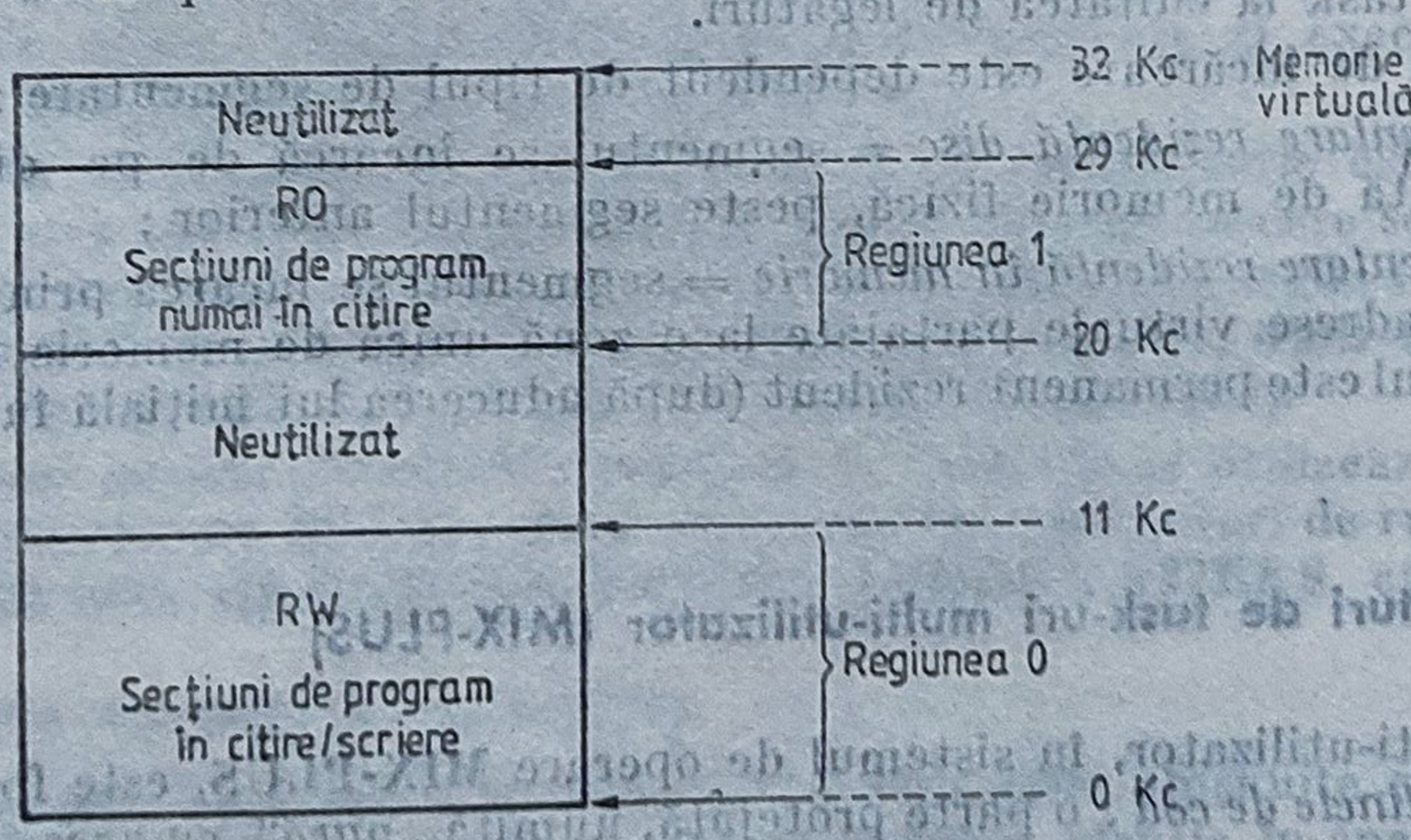


Fig. 5.12. Structură de task multi-utilizator

Pentru maparea unui task multi-utilizator obișnuit (fără spații **I** și **D** separate), Editorul de legături alocă doi descriptori de segmente (ferestre de adrese) virtuale, inițializate astfel :

— **WDB 0** = descrie domeniul adreselor virtuale pentru porțiunea „impură" (**RW**) a imaginii task ;

— **WDB 1** = descrie domeniul adreselor virtuale pentru porțiunea „pură" (**RO**) a imaginii task.

Pentru minicalculatoarele cu spații de instrucțiuni (**I**) și date (**D**) separate, se pot de asemenea crea task-uri multi-utilizator. Deși teoria generală, prezentată anterior (vezi paragraful 5.12.1), rămîne valabilă, cele două regiuni ale unui task multi-utilizator (**RO** și **RW**) se împarte fiecare în zone separate, mapate în spațiile de instrucțiuni (**I**) și date (**D**).

Pentru maparea unui task multi-utilizator cu spații **I** și **D** separate (/MU/ID), Editorul de legături alocă patru descriptori de segmente (ferestre de adrese) virtuale, inițializate astfel :

— **WDB 0** = descrie domeniul adreselor virtuale pentru zona de instrucțiuni (**I**) a porțiunii „impure" (**RW**) ;

— **WDB 1** = descrie domeniul adreselor virtuale pentru zona de date (**D**) a porțiunii „impure" (**RW**) ;

— **WDB 2** = descrie domeniul adreselor virtuale pentru zona de instrucțiuni (**I**) a porțiunii „pure" (**RO**)

— **WDB 3** = descrie domeniul adreselor virtuale pentru zona de date (**D**) a porțiunii „pure" (**RO**).

Separarea atributelor de date (**D**) și instrucțiuni (**I**) ale porțiunilor „impure" și „pure" se face la editarea de legături conform descrierilor de secțiune program (**.PSECT**). Dimensiunea totală a unui task multi-utilizator este în acest caz de 64 kcuvinte (32 kc pentru spațiul **I** și 32 kc pentru spațiul **D**).

5.12.4. Structura unui task pe suport extern

În urma editării de legături se obține o imagine task, pe un suport extern **MIX** (disc), ce conține și toate structurile de date necesare identificării task-ului de alte componente sistem. Monitorul (responsabil cu alocarea resurselor sistem) trebuie să aibă acces la toate informațiile asociate tuturor task-urilor în sistem. Trebuie să știe, de exemplu, dimensiunea unui task și prioritatea sa, precum și modul în care acesta intenționează să utilizeze sistemul de operare. Editorul de legături aloçă spațiu în imaginea task pentru structurile de date solicitate de Monitor — așa numita zonă de *„antet" (header)* — pe care o și inițializează conform opțiunilor implicite sau explicite, solicitate de utilizator.

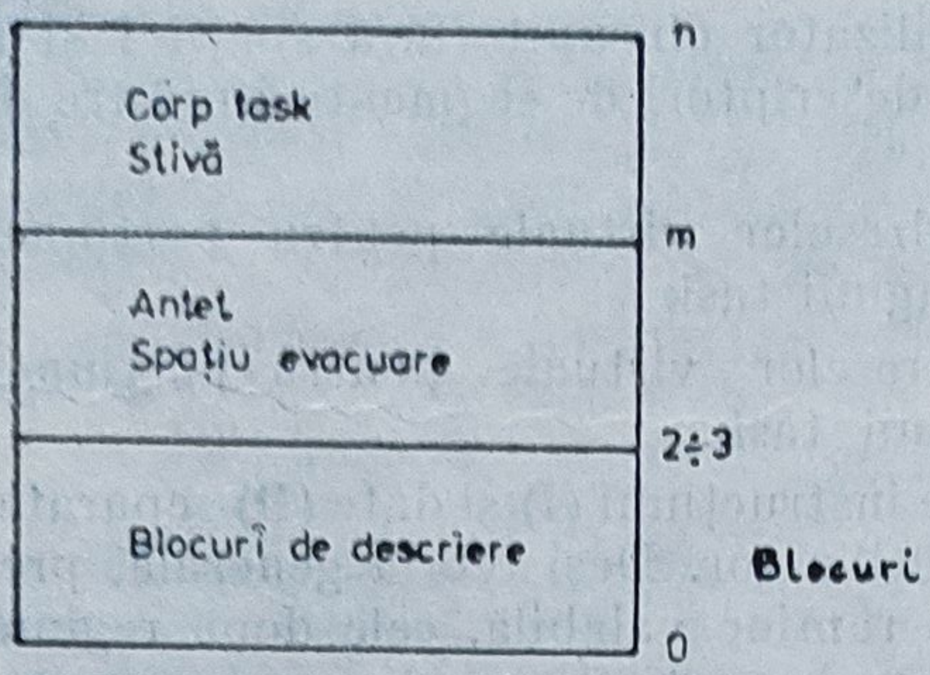
Într-o formă simplificată, imaginea unui task pe disc constă din următoarele părți, fizic contigue :

— Bloc(uri) de descriere task *(label)* ;

— Antet task *(header)* ;

— Stivă task *(stack)* ;

— Corp task.



Fgi. 5.13. Imagine simplificată a unui task pe un suport extern (disc)

Figura 5.13 ilustrează structura simplificată a unei imagini task pe suport extern (disc).

Unitatea de alocare a unei imagini de task este blocul (256 cuvinte), indiferent de tipul de suport extern (bandă perforată, bandă magnetică, casetă magnetică, disc).

Bloc(urile) de descriere task, nu sînt încărcate în memorie și conțin *date de identificare a task-ului*, după cum urmează :

Blocul 0:

— nume task ;
— parametrii partiției atașate task ;
— parametrii generali task ;
— data de creare task ;
— descriptorii bibliotecilor partajate sau zonelor de comun la care se conectează task-ul ;
— atribute task (dimensiune, prioritate, etc.) ;
— alte informații utilizate la instalare.

Blocul 1 (și eventual 2) :

— tabela de atribuiri numere logice — dispozitive de intrare/ieșire (**LUN** < 128).

În cazul în care numărul de **LUN**-uri depășește 128 se alocă un nou bloc de descriere a atribuirilor (maximum 255).

Blocul 3 (numai în cazul structurilor segmentate) :

— lista de încărcare a segmentelor.

Informațiile conținute în bloc(urile) de descriere task sînt folosite la instalare (comanda operator **MCL/DCL INS**tall) pentru crearea blocului de control asociat (**TCB**) și pentru inițializarea antetului rezident al task-ului.

Antetul de task, urmează imediat blocurilor de descriere task și este cadrat la multiplu de bloc. Task-ul este încărcat în memorie începînd cu antetul de task.

Acesta este divizat în două părți :

— o parte fixă ;
— o parte variabilă.

Partea fixă conține informații de descriere și referință utilizate pe durata execuției task-ului, cum sînt :

— codul de identificare al utilizatorului (**UIC**) ;
— valorile inițiale ale **PC**, **PS** și **SP** ;
— descriptorii vectorilor de tratare **SST** ai sistemelor de depanare și task ;
— referințe către diferite rutine de tratare **AST** ;
— referințe către diferite zone de reentranță task ;
— alte informații de salvare sau utilizate la instalarea task-ului în memorie.

Partea variabilă conține :
— blocurile de descriere a segmentelor virtuale ;
— tabela de unități logice ;
— contextul de salvare task.

În cazul anumitor imagini de task, între blocurile) de descriere și antetul de task există o zonă suplimentară, numită *zonă de evacuare a task-ului*, egală cu lungimea celei mai lungi ramuri din structura de segmentare a task-ului (sau egală cu lungimea totală a task-ului în cazul unei structuri contigue).

În momentul fixării în memorie a unei astfel de structuri se ignoră zona de evacuare atașată task-ului.

Antetul de task conține informații utilizate de Monitor pe durata execuției task-ului. El este creat și parțial inițializat de Editorul de legături ; la instalare se inițializează restul informațiilor din antetul de task.

Zona de antet din memorie este întotdeauna accesibilă task-ului utilizator. În sistemele cu relocare, din motive de protecție, Monitorul duplică antetul de task în zona de memorie sistem cu alocare dinamică (**DSR**).

Pe sistemul de operare **MIX-PLUS** există posibilitatea (implicită a) duplicării antetului de task direct în partiția utilizator (așa numita opțiune de „*antet extern*“ la editarea de legături : **/XH** sau instalare : **/XHR = YES**), în scopul economisirii memoriei sistem cu alocare dinamică. Această zonă este protejată față de restul partiției și se include, la evacuare, în zona de evacuare a task-ului.

Stiva task-ului urmează antetului de task și va fi utilizată, de către task și sistem, ca zonă de salvare, transmitere parametri sau reentranță.

Dimensiunea stivei se stabilește la editarea de legături a task-ului, implicit (512 octeți) sau explicit (opțiunea **STACK**).

De remarcat faptul că task-ul utilizator nu trebuie să-și aloce stiva în mod explicit (prin rezervări) și nici să inițializeze registrul de stivă (**SP**). Aceste operații sînt realizate de către Editorul de legături.

Corpul taskui-ul urmează stivei și conține datele și instrucțiunile utilizator.

În cazul unei structuri segmentate (rezidente disc sau memorie), aceste informații (suficiente pentru structurile de task contigue) sînt completate de Editorul de legături cu (vezi fig. 5.14):
— tabele de descriere ale segmentelor ;
— vectorii de autoîncărcare ;
— segmentele aparținînd task-ului.

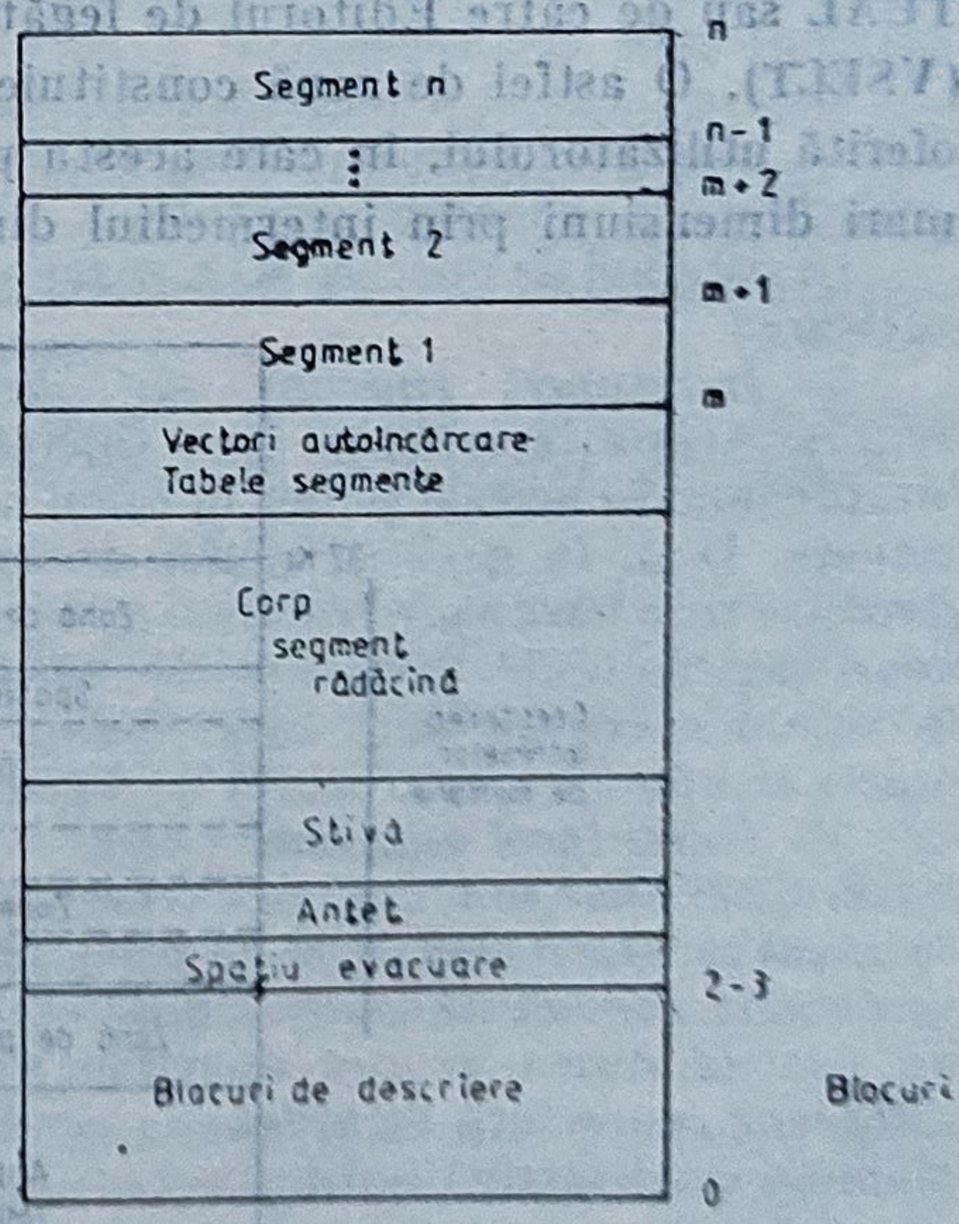


Fig. 5.14. Structura de segmente rezidentă în memorie a unui task pe suport extern (disc)

Spre deosebire de task-urile mono-utilizator (cu structură contiguă sau segmentată), imaginea disc a task-urilor multi-utilizator este diferită. Porțiunea de task cu acces numai în citire (RO) este plasată în partea finală (sfîrșitul) imaginii disc. Adresa relativă a acestei porțiuni (numărul relativ de bloc la începutul imaginii disc) și lungimea asociată (în număr de blocuri), sînt păstrate în primul bloc de descriere a task-ului (*label*), urmînd a fi folosite la instalare și încărcare în memorie.

5.12.5. Structura unui task în memorie

Imaginea memorie a unui task este acea parte a task-ului, încărcată de către sisteme în memoria fizică la execuție (**MIX/MIX-PLUS**) sau instalare cu fixare (**MIX/MIX-RT**).

Zona blocurilor de descriere task și zona de evacuare nu sînt încărcate în memorie.

După încărcarea în memorie a unui task, memoria atașată acestuia poate fi împărțită (în sistemele cu relocare), în mai multe zone contigue:

— *Zona antetului extern* (pe sistemele **MIX-PLUS**): specificată opțional, la editarea de legături sau instalare, ea duplică antetul de task și este folosită de Monitor pe durata execuției task-ului;

— *Zona de program virtuală*: este creată de utilizator prin opțiunea **VIRTUAL** sau de către Editorul de legături la declararea unor secțiuni virtuale (**VSECT**). O astfel de zonă constituie o posibilitate de alocare suplimentară oferită utilizatorului, în care acesta poate crea și referi structuri de date de mari dimensiuni prin intermediul directivelor de gestiune a memoriei.

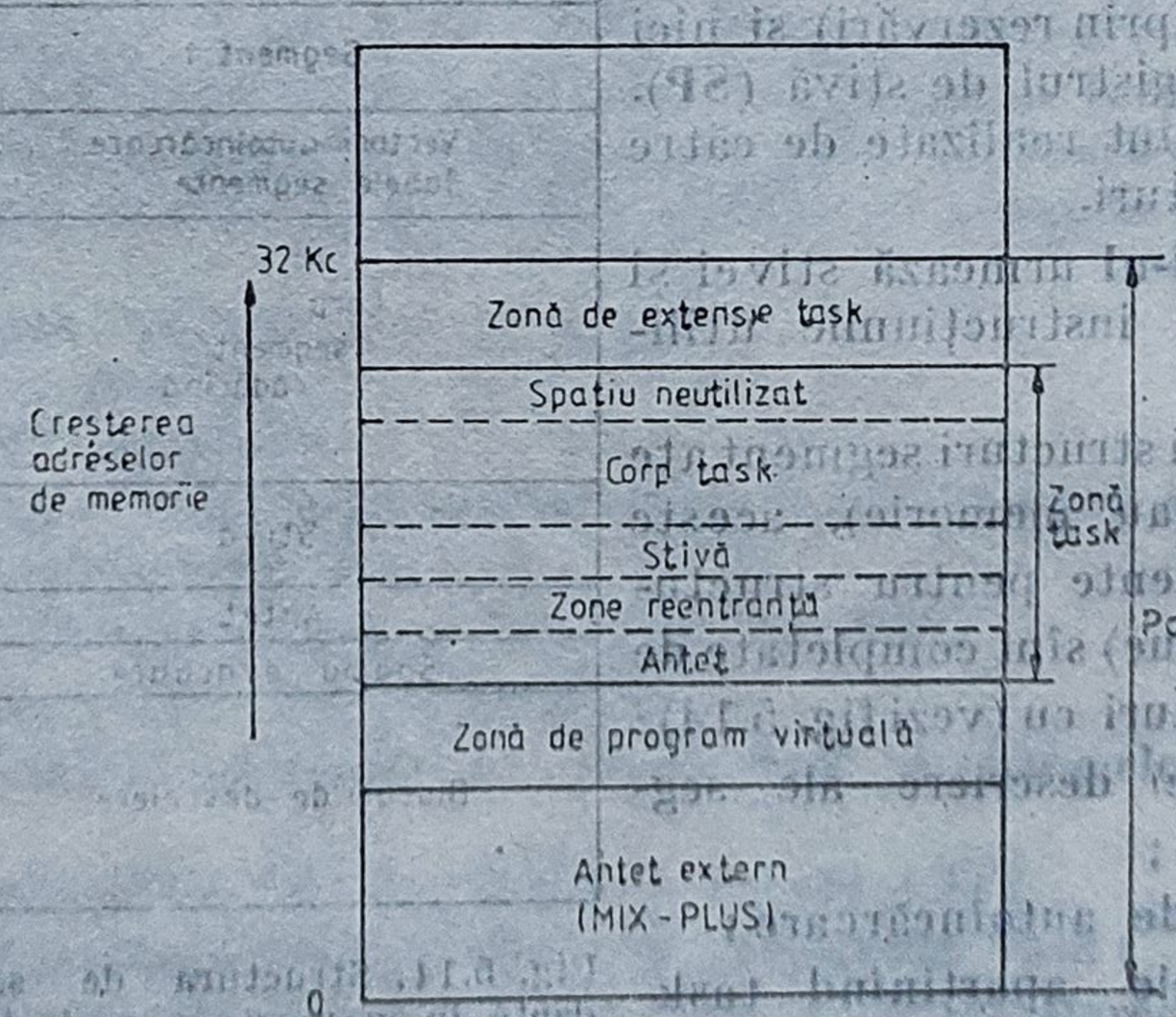


Fig. 5.15. Structura unui task în memeor

— *zona task* : este formată din
— antet task (imagine identică cu cea de pe disc) ;
— zone reentranță ;
— stivă ;
— corp de task (inclusiv, în cazul structurilor segmentate, tabelele de descriere ale segmentelor, vectorii de autoîncărcare și, fie spațiu pentru cea mai lungă ramură din structura de segmente — segmentare rezidentă disc, fie totalitatea segmentelor taskului — segmentare rezidentă în memorie) ;
— spațiu neutilizat, disponibil în partiție.

— *Zona de extensie task* : aceasta constituie o zonă de memorie suplimentară alocată dinamic, la instalarea unui task (cuvîntul cheie /**INC**) sau în timpul execuției task-ului (directiva **EXTK$**).

În cazul sistemelor fără relocare, este prezentă numai zona task.

Figura 5.15. prezintă, schematic, structura unui task în memorie.

6

Excepții și întreruperi software

6.1. Apariția întreruperilor software

In timpul execuției unui task pot apare condiții așteptate sau neașteptate ce provoacă întreruperea acestuia. Apariția unor astfel de condiții este cunoscută și sub numele de *excepții* sau *întreruperi software*.

Sistemul de operare **MIX** permite, la apariția unei întreruperi software, o tratare a condițiilor de excepție și la nivelul task-urilor utilizator dînd posibilitatea acestora să reacționeze prompt la apariția sau terminarea (completarea) unor evenimente.

În acest caz se transferă controlul unor rutine aparținînd task-ului utilizator, asociate întreruperilor software, care tratează în mod particular condițiilor de excepție apărute.

În sistemul de operare **MIX** se recunosc două tipuri distincte de întreruperi software :

— *Întreruperi sincrone* sau *derutări* (**SST** — *Synchronous Software Traps*). Aceste întreruperi apar sincron cu execuția unui task, producîndu-se întotdeauna în același punct în cazul repetării secvenței de program corespunzătoare. Deși, în general, astfel de întreruperi reprezintă condiții de eroare generate prin hardware (condiții așteptate), ele se produc și în cazul executării unor instrucțiuni. Întreruperile sincrone reprezintă evenimente obișnuite în cursul execuției unui task, acesta putînd prelua controlul la momentul precis al apariției unui astfel de eveniment.

— *Întreruperi asincrone* (**AST** — *Asynchronous Software Traps*). Aceste întreruperi apar asincron cu execuția unui task, producîndu-se în alte puncte în cazul repetării secvenței de program corespunzătoare. Ele sînt asociate întotdeauna unor evenimente semnificative din sistem, task-ul implicat neavînd un control direct asupra apariției unui astfel de eveniment. Întreruperile asincrone reprezintă întotdeauna condiții neașteptate, cum ar fi de pildă terminarea unei intrări/ieșiri.

Pentru a stabili puncte de intrare specifice pentru diferitele tipuri de întreruperi software e necesară utilizarea unor directive sistem corespunzătoare.

Punctele de intrare pentru întreruperile sincrone (SST) sînt colectate într-o tabelă unică per task, iar cele pentru întreruperile asincrone (AST) sînt specificate individual pentru fiecare tip de AST (prin directive Monitor). În cazul apariţiei unei întreruperi software, Monitorul transferă controlul rutinei de tratare asociate (dacă s-a specificat un punct de intrare corespunzător).

6.2. Întreruperi software sincrone (SST)

Aceste condiţii sincrone sînt generate în următoarele situaţii :

— erori de adresare la memorie (adresă impară, adresă inexistentă, erori pe BUS, time-out, etc.), forţîndu-se o derută hardware la locaţia de memorie 4 ; la apariţia unor violări de limite stivă, se forţează întotdeauna o eroare sistem ;

— apariţia unei erori de paritate la memorie, forţîndu-se o derută hardware la locaţia de memorie 114 ; dacă acestea apar în Monitor, drivere sau, task-uri privilegiate mapate la Monitor se forţează întotdeauna o eroare sistem şi se afişează, pe consola operator, următorul mesaj :

****MIX MEMORY PARITY ERROR — MONITOR STOP****

Dacă aceasta apare într-un task utilizator, se încearcă întotdeauna o tratare de tip **AST**, şi nu **SST** ;

— violări de acces la memorie (acces la zone de memorie protejate), forţîndu-se o derută hardware la locaţia de memorie 250 ; dacă aceasta apare în Monitor sau o rutină sistem, se forţează întotdeauna o eroare sistem ;

— derută ocazionată de poziţionarea bitului T în cuvîntul de stare program (**PS**) sau de execuţia instrucţiunii **BPT**, forţîndu-se o derută hardware la locaţia de memorie 14 ; dacă aceasta apare în Monitor, se forţează întotdeaana o eroare sistem, dîndu-se controlul subsistemului de depanare **MDS** (dacă acesta a fost inclus în sistem) ;

— execuţia instrucţiunii **IOT**, forţîndu-se o derută hardware la locaţia de memorie 20 ; dacă aceasta apare în Monitor, se forţează întotdeauna o eroare sistem ;

— execuţia unei instrucţiuni rezervate (referitoare la modul de execuţie curent) sau instrucţiuni ilegale (neimplementate de Unitatea Centrală), forţîndu-se o derută hardware la locaţia de memorie 10 ; dacă aceasta apare în Monitor, se forţează întotdeauna o eroare sistem ;

— execuţia unei instrucţiuni **EMT** neimplementată în sistemul de operare **MIX**, forţîndu-se o derută hardware la locaţia de memorie 30. Dacă instrucţiunea este **EMT 377** (sau **EMT 376** pentru task-urile privilegiate mapate la Monitor), se transferă controlul dispecerului de directive Monitor ; în caz contrar se încearcă o tratare **SST** ;

— execuţia instrucţiunii **TRAP**, forţîndu-se o derută hardware la locaţia de memorie 34. Dacă instrucţiunea se execută în modul sistem (*„Kernel“*), controlul este transferat unei rutine Monitor de setare a conţinutului **$DSW** cu valoarea specificată de instrucţiunea **TRAP** ; în caz contrar se încearcă o tratare **SST** ;

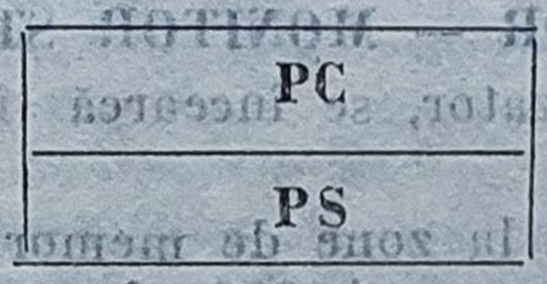
— execuția unei instrucțiuni de virgulă mobilă, aparținînd setului restrîns (**FIS**) sau extins (**FPP**) de instrucțiuni de virgulă mobilă, forțîndu-se o derută hardware la locația de memorie 244. Derută apare numai la acele calculatoare care nu au implementat :

— setul restrîns de instrucțiuni de virgulă mobilă (**FIS**) (**INDEPENDENT I-100**, la care acesta este opțional și pentru care se încearcă întotdeauna o tratare de tip **SST**), sau

— setul extins de instrucțiuni de virgulă mobilă (**FPP**) (**INDEPENDENT I-100, CORAL 4001(A), 4011(A), 4030** și pentru care se încearcă întotdeauna o tratare de tip **AST**, și nu **SST**).

În cazul în care se dorește o tratare corespunzătoare a unora din aceste întreruperi software, sincrone, se prezintă sistemului un vector de tratare **SST**, conținînd o intrare pentru fiecare întrerupere software (7 intrări, într-o ordine apropiată de cea prezentată anterior). Fiecare intrare reprezintă adresa unei rutine de tratare a întreruperii software corespunzătoare. În cazul în care nu se dorește o tratare utilizator, se specifică o intrare vidă (zero).

La apariția unei întreruperi software sincrone, Monitorul transferă controlul rutinei corespunzătoare tipului de întrerupere, plasînd conținutul Stării program (**PS**) și a Contorului de adrese (**PC**) în stiva asociată task-ului :



În funcție de sursa întreruperii sincrone, în stivă se pot memora informații adiționale, așa cum se întîmplă în următoarele două cazuri :

a) violări de acces la memorie :

SP + 10 = **PS**
SP + 06 = **PC**
SP + 04 = Registrul de stare protecție memorie (**SR0**)
SP + 02 = **PC**-ul virtual al instrucțiunii eronate (**SR2**)
SP + 00 = Registrul de stare instrucțiune (**SR1**)

b) execuția instrucțiunii **TRAP** sau **EMT** (diferită de 377 sau 376 — pentru task-urile nemapate sau task-urile privilegiate mapate) :

SP + 04 = **PS**
SP + 02 = **PC**
SP + 00 = Operandul instrucțiunii (octetul din dreapta) multiplicat cu 2 (fără extensie de semn).

Cu excepția **PS** și **PC**, informațiile adiționale trebuie eliminate din stivă înaintea ieșirii din rutina de tratare **SST**.

După prelucrarea condiției de excepție, se redă controlul programului întrerupt prin execuția unei instrucțiuni **RTI** sau **RTT** (retur din secvența de întrerupere). La apariția întreruperii nu se salvează registrele generale **R0** ÷ **R6**; în cazul în care rutina de tratare utilizează unele din aceste registre, salvarea și refacerea lor rămîne în sarcina task-ului utilizator.

Execuția unei rutine de tratare **SST** nu diferă cu nimic de execuția normală a task-ului. Se pot utiliza toate serviciile Monitor, se poate modifica conținutul cuvîntului de stare al directivei (**$DSW**), se pot modifica registrele

generale, se pot șterge sau poziționa indicatori de evenimente ; aceste acțiuni rămîn efective și după redarea controlului secvenței întrerupte. Mai mult, în cadrul rutinei de tratare **SST** se poate înlocui sau elimina contextul de întrerupere în stivă (**PS** și **PC**) cu un altul sau se poate transfera direct controlul în orice parte a task-ului utilizator.

În general, e necesar ca rutinele de tratare **SST** să fie reentrante întrucît pot apare alte întreruperi software în cadrul acestor rutine. Această situație e normală, neputîndu-se distinge execuția rutinei de tratare **SST** față de execuția obișnuită a task-ului utilizator. O astfel de rutină poate fi întreruptă de către o altă condiție de excepție sincronă (**SST**) sau asincronă (**AST**)

În momentul apariției unei întreruperi software, **SST**, pentru care există o intrare validă în vectorul corespunzător de tratare a excepțiilor, rutina de tratare **SST** se execută în același mod *(Utilizator* sau *Supervizor)* cu cel din momentul specificării vectorului **SST** (prin execuția directivelor Monitor **SVDB$** sau **SVTK$**). Dacă punctul de intrare constă dintr-o adresă invalidă impară sau în afara spațiului virtual de adresare), la transferul controlului, de către Monitor la această adresă, poate apare o altă întrerupere software. Noua rutină de tratarea specifică **SST** se va executa întotdeauna în modul *Utilizator*.

Fiecărui task îi sînt asociați doi vectori de tratare a excepțiilor cu descriptori localizați în antetul de task *(header)* :

— un *vector primar de excepție*, asociat sistemelor de depanare furnizate de **MIX** sau de către utilizator (prin programarea **SVDB$** sau **ODTV**) ;

— un *vector secundar de excepții*, asociat task-ului utilizator (prin programarea **SVTK$** sau **TSKV**).

Fiecare din acești vectori este format din maximum opt adrese de tratare **SST**, avînd următoarea organizare :

Structură vector :	**Adresă octală :**	**Vector hardware asociat :**
erori adresare memorie	0	
violări de acces memorie	2	4
derută bit T sau execuție **BPT**	4	250
execuție **IOT**	6	14
instrucțiunea rezervată/ilegală	10	20
execuție **EMT**	12	10
execuție **TRAP**	14	30
execuție instrucțiune virgulă flotantă	16	34
	20	244

Fiecare vector de tratare este prezentat sistemului printr-un descriptor asociat.

Fiecare *descriptor*, cu o lungime de 2 cuvinte, are următoarea structură :

— *primul cuvînt* conține *adresa vectorului de excepție*. În cazul în care acest cuvînt este zero, nu există o tratare de condiții asociată :

— cel de-al *doilea cuvînt* conţine *numărul de condiţii de excepţie* ce urmează a fi tratate (numărul de puncte de intrare din vectorul de excepţie).

Sistemul de operare **MIX** pune la dispoziţia utilizatorilor *două metode pentru specificarea tratărilor de excepţie sincrone:*

— prin *specificarea statică a adreselor vectorilor de excepţie.* Cea mai utilizată, permiţînd specificarea adreselor încă de la editarea de legături (opţiunile **ODTV** şi **TSKV**)

— prin *specificarea sau eliminarea dinamică a adreselor vectorilor de excepţie*, utilizînd directivele Monitor **SVDB$** şi **SVTK$**.

În momentul apariţiei unei condiţii de excepţie, dispecerul sistem de excepţii, nu diferenţiază condiţiile de excepţie apărute, transferînd controlul la primul vector de tratare excepţii valid existent. Dacă vectorul primar nu este specificat, se încearcă trecerea controlului către vectorul secundar. Dacă nici acesta nu există, sau nu se specifică intrarea asociată condiţiei de excepţie, se transferă controlul rutinelor sistem de tratare a condiţiilor de excepţie, transformînd tratarea de excepţie în tratare de terminare anormală a taskului implicat.

6.3. Întreruperi software asincrone (AST)

Scopul principal al generării unei întreruperi software asincrone este cel al informării task-ului implicat despre apariţia unui eveniment semnificativ.

De exemplu, sistemul validează apariţia unei întreruperi **AST** asociate unui eveniment (cum ar fi terminarea unei operaţii de intrare/ieşire) şi o transmite task-ului utilizator în momentul apariţiei acesteia, prin lansarea în execuţie a unei rutine utilizator de tratare a întreruperii. La terminarea tratării utilizator, se redă controlul secvenţei întrerupte.

Întrucît aceste întreruperi apar asincron, relativ la execuţia unui task, mecanismul de întrerupere este şi el asincron, task-ul neputînd avea un control direct la momentul precis al generării întreruperii software.

În unele cazuri, task-urile utilizator pot utiliza simultan două mecanisme de informare : indicatori de evenimente şi întreruperi **AST** (prin utilizarea anumitor directive Monitor). Această posibilitate permite utilizatorului specificarea unei aceleiaşi rutine de tratare **AST** pentru mai multe directive, fiecare identificată de către un indicator de eveniment diferit. La apariţia unei întreruperi **AST**, indicatorul de eveniment specificat (setat de către Monitor) determină, în mod unic, tipul de acţiune utilizator ulterioară.

În opoziţie cu execuţia unei rutine de tratare **SST**, ce nu poate fi distinsă de execuţia normală a task-ului întrerupt, rutinele de tratare AST se execută cu modul de acces existent în momentul declarării acestei tratări, iar contextul asociat fiecărei tratări va fi depus în starea corespunzătoare acestui mod de acces. O rutină de tratare AST poate fi întreruptă de o condiţie de excepţie **SST**, dar niciodată de către o altă rutină de tratare **AST**.

6.3.1. **Prelucrarea întreruperilor AST**

În momentul în care se solicită o tratare AST asociată unui task, se execută următoarele acțiuni :

a) sistemul plasează cererea de întrerupere **AST** într-o coadă atașată fiecărui bloc de descriere al task-ului (precizată de locațiile **T.ASTL/T.ASTL + + 2** din **TCB**) ;

b) în momentul în care condițiile o permit, întreruperea **AST** este plasată task-ului utilizator ;

c) rutina de tratare **AST** asociată cererii de întrerupere primește controlul.

Sistemul de operare **MIX** menține cîte o coadă de tip **FIFO** (primul introdus, primul extras), ordonată după modul de acces, pentru fiecare tip de tratare specificat de un task din sistem.

În momentul apariției unei cereri de întrerupere **AST**, aceasta este transmisă direct task-ului implicat sau plasată în coada de cereri **AST** în funcție de starea task-ului și a biților de control din **TCB**.

În cazul în care task-ul este rezident în memorie, este activ și tratarea **AST** este validată, execuția task-ului este întreruptă și se transmite direct controlul rutinei de tratare corespunzătoare.

În cazul în care o întrerupere de tip **AST** apare în timpul execuției unei rutine de tratare **AST**, ultima întrerupere este plasată automat în coada de tratare **AST** și prelucrarea următorului **AST** din coadă are loc la terminarea prelucrării curente **AST** (cu condiția ca recunoașterea **AST** să nu fi fost invalidată de rutina de tratare **AST**).

Dacă un task este suspendat la apariția unei întreruperi **AST**, task-ul rămîne suspendat și după terminarea rutinei de tratare AST, cu excepția cazului în care relansarea task-ului suspendat a fost cerută de rutina de tratare **AST** sau de către un alt task (prin execuția directivei **RSUM$** sau comenzii operator **MCL RESume** sau **DCL CONTINUE**).

Dacă un task este stopat la apariția unei întreruperi **AST**, Monitorul destopează task-ul pe durata execuției rutinei de tratare AST și îl stopează din nou la ieșirea din tratarea **AST**, cu excepția cazului în care se forțează destoparea task-ului în rutina de tratare **AST** sau de către un alt task (prin execuția directivei **USTP$** sau comenzii operator **MCL RESume** sau **DCL CONTINUE**).

Dacă la apariția unei întreruperi **AST**, un task este blocat sau stopat în așteptarea completării unui eveniment, diferit de cel asociat directivei la specificarea rutinei de tratare **AST**, task-ul rămîne blocat sau stopat și după executarea rutinei de tratare AST, cu excepția cazului în care indicatorul de eveniment asociat a fost poziționat de către rutina de tratare **AST**, de către un alt task, sau de completarea acțiunii asociate evenimentului.

Dacă la apariția unei întreruperi **AST** un task era evacuat (local sau la distanță), Monitorul introduce (FIFO) cererea AST în coada specifică, forțează aducerea task-ului în memorie și reactivează rutina de tratare **AST** după încărcarea completă a task-ului în memorie.

În mod similar, la readucerea unui task în memorie, anterior evacuat Monitorul forțează execuția rutinelor de tratare **AST** asociate recepției de

mesaje, dacă coada de recepție asociată task-ului conține una sau mai multe intrări. Această metodă previne pierderea întreruperilor **AST** asociate recepției de mesaje pentru task-urile evacuate.

În cazul introducerii de date buferate de la terminal, Monitorul stopează execuția task-ului evacuabil ce a solicitat această operație și îl evacuează pînă la terminarea efectivă a introducerii de date (notificată Monitorului de către driverul de terminal). Task-ul își reia execuția după terminarea introducerii de date. Pe durata stopării, task-ul poate executa o rutină de tratare AST la apariția acesteia ; el rămîne însă în continuare stopat, după ieșirea din tratarea **AST**, cu excepția cazului terminării în paralel a introducerii de date (rutina AST poate reactiva task-ul stopat numai prin forțarea terminării anormale a introducerii de date — **IO.KIL**).

Tratarea întreruperilor de tip **AST** poate fi permisă sau interzisă dinamic prin utilizarea unor directive Monitor (**ENAR$S**, respectiv **DSAR$S**). Utilizarea acestor directive este oportună atunci cînd se dorește protejarea execuției unor porțiuni critice (ce pot conține informații modificabile de o rutină de tratare **AST**, de exemplu). În acest caz, la apariția unor întreruperi **AST**, dacă recunoașterea acestora a fost invalidată, cererile de tratare sînt plasate în coada asociată task-ului (**FIFO**) și luate în considerare la revalidarea tratării **AST**.

În cazul în care nu se dorește o tratare **AST** pentru anumite tipuri de întreruperi **AST**, sistemul ignoră apariția acestora.

În momentul transmiterii controlului către rutina de tratare **AST**, se plasează în stiva asociată task-ului întrerupt un context complet privind starea acestuia, pentru a permite rutinei de tratare un acces deplin la toate serviciile Monitor. Acest context e format din :

— masca indicatorilor de așteptare (**EFMW**) ;
— cuvîntul de stare program (**PS**) anterior **AST** ;
— contorul de adrese (**PC**) anterior **AST** ;
— cuvîntul de stare al directivei (**DSW**) atașat task-ului, anterior **AST**.

Masca indicatorilor de așteptare permite rutinei de tratare AST stabilirea unor condiții necesare deblocării task-ului întrerupt. Valoarea octală a măștii indicatorilor de așteptare se obține din locația **T.EFLM** din **TCB** sau **H.EFLM** din antetul de task *(header)* curent. Valoarea acestei măști și gama indicatorilor de evenimente implicați (conținută în locația **T.FFLM + 2**/**H.EFLM + 2**), corespunde ultimei directive Monitor **WTSE$ (WTLO$)** sau **STSE$ (STLO$)**, executată de către task-ul curent. În cazul în care task-ul nu a utilizat directive de blocare (așteptare), anterior tratării **AST**, masca indicatorilor de așteptare este fără semnificație. În cazul în care masca indicatorilor de așteptare este validă, modificarea acesteia în rutina AST poate duce la apariția unor fenomene de așteptare (blocare) nedefinite.

În funcție de tipul întreruperii **AST**, stiva poate conține și alți parametri adiționali. În momentul pasării controlului rutinei de tratare **AST**, registrele generale **R0 ÷ R6** nu sînt salvate ; în cazul în care rutina de tratare le utilizează, salvarea și refacerea registrelor cade în sarcina acesteia.

La terminarea tratării **AST**, trebuie eliminați în mod explicit parametrii adiționali, dependenți de tipul de întrerupere asincronă (de sus în jos, pînă la cuvîntul de stare al directivei). Ieșirea dintr-o rutină de tratare **AST** se face prin programarea unei directive Monitor speciale (**ASTX$S**). Această directivă pasează controlul :

— unei alte rutine de tratare **AST** (în cazul în care coada de elemente **AST** asociată task-ului conține și alte intrări) ;

— secvenței întrerupte (în cazul în care coada asociată este vidă).

6.3.2. Tipuri de tratări AST

Tipurile de tratări **AST** permise unui task utilizator sînt următoarele :

— *Recuperare din avarie :* Dacă această tratare este validată prin directiva Monitor **SPRA$** și task-ul care a solicitat-o nu este evacuat, se forțează o întrerupere **AST** la ieșirea din avarie. Orice altă recuperare din avarie ulterioară nu va fi luată în considerare pînă la terminarea tratării curente.

— *Excepție virgulă mobilă :* Dacă această tratare este validată prin directiva Monitor **SFPA$**, se forțează o întrerupere **AST** la apariția unei excepții virgulă mobilă, în cazul implementării setului extins de instrucțiuni de virgulă mobilă (**FPP**). Orice altă excepție virgulă mobilă ulterioară nu va fi luată în considerație pînă la terminarea tratării curente.

— *Recepția mesaj sau referință la o regiune (zonă de comun) :* Dacă această tratare este validată prin directiva Monitor **SRDA$**, respectiv **SRRA$**, se forțează o întrerupere **AST** la recepționarea unui mesaj sau unei referințe la o regiune (zonă de comun). Orice alte tratări **AST** de recepție ulterioare nu vor fi luate în considerare decît după terminarea tratării curente.

— *Expirarea unui interval de timp specificat :* Această tratare este validată prin specificarea parametrului **AST** la apelul directivei Monitor **MRKT$**, forțîndu-se o întrerupere **AST** la epuizarea intervalului de timp indicat.

— *Terminarea unei operații de intrare/ieșire :* Această tratare este validată prin specificarea parametrului **AST** la apelul directivei Monitor **QIO$**, respectiv **QIOW$**, forțîndu-se o întrerupere **AST** la terminarea operației de intrare/ieșire solicitate.

— *Eroare de paritate :* Dacă această tratare este validată prin directiva Monitor **SPEA$**, se forțează o întrerupere **AST** la apariția unei erori de paritate la memorie.

— *Terminare anormală task :* Dacă această tratare este validată prin directiva Monitor **SREA$**, se forțează o întrerupere **AST** la terminarea anormală a unui task, ca urmare a execuției unei directive **ABRT$** sau comenzi operator **MCL/DCL ABO**rt.

— *Terminare anormală condiționată task :* Dacă această tratare este validată prin directiva Monitor **SREX$**, se forțează o întrerupere **AST** la terminarea anormală condiționată a unui task, ca urmare a execuției unei directive **ABRT$** sau comenzi operator **MCL/DCL ABO**rt.

— *Recepți unei linii de comandă pentru un task de tip* **CLI** : Dacă această tratare este validată prin directiva Monitor **SCAA$**, se forțează o întrerupere **AST** la recepționarea unei linii de comandă de către un task de tip **CLI**.

— *Returnarea stării unui task „fiu"* : Această tratare este validată prin specificarea parametrului AST la apelul directivelor **SPWN$, CNCT$** sau **SDRC$** (solicitînd conectarea unui task „tată" la un task „fiu"), forțîndu-se o întrerupere **AST** către task-ul (urile) „tată" conectate, la terminarea execuției sau emiterea de stare a unui task „fiu".

— *Completarea unei introduceri de date nesolicitate :* Această tratare este validată prin specificarea parametrului **AST** la apelul funcției **IO.ATA** către driver-ul de terminal full-duplex, forțîndu-se o întrerupere **AST** la completarea (terminarea) unei introduceri de date nesolicitate de la terminal.

— *Atașarea sau detașarea unui terminal virtual :* Această tratare (existentă numai pe sistemul **MIX-PLUS**), este validată prin specificarea parametrului **AST** la apelul directivei Monitor **CRVT$**, forțîndu-se cîte o întrerupere **AST** specifică la atașarea, respectiv detașarea, unui nou terminal virtual.

— *Efectuarea unei operațiuni de intrare/ieșire date (de) pe un terminal virtual :* Această tratare (existentă numai pe sistemul **MIX-PLUS**), este validată prin specificarea parametrilor **AST** (intrare și ieșire) la apelul directivei Monitor **CRVT$**, forțîndu-se cîte o întrerupere **AST** după efectuarea unei operațiuni de introducere/extragere date (de) pe un terminal virtual.

Fiecărui tip de întrerupere **AST** îi este asociat un bloc de descriere, creat în mod dinamic de către sistem, ce conține printre altele contextul de salvare, și care este introdus în coada de tratare AST corespunzătoare fiecărui task.

Tratările de tip AST : recuperare din avarie, excepție virgulă mobilă, recepție mesaj și recepție referință, eroare de paritate, terminare anormală task (condiționată sau nu), recepție linie de comandă, pot fi permise sau interzise prin crearea (respectiv ștergerea) unor intrări în antet-ul asociat fiecărui task.

În momentul apariției simultane a mai multor întreruperi de tip **AST** ordinea de lansare în execuție a rutinelor de tratare AST este aproximativ cea prezentată la tipurile de tratări **AST**. Cu excepția întreruperii **AST** recuperare din avarie, toate celelalte tipuri de întreruperi **AST** sînt locale unui task. În cazul unei recuperări din avarie, se forțează o tratare de tip **AST** pentru toate task-urile din sistem care au validat tratarea corespunzătoare.

Un caz special de utilizare a tratării de tip **AST** îl constituie poziționarea stării de intrare/ieșire la terminarea unei operații de intrare/ieșire. Utilizînd mecanismul de întreruperi asincrone, poziționarea stării de intrare/ieșire (în cazul specificării unui bloc de stare al intrării/ieșirii (**IOSB**)) este efectuată în contextul task-ului care a inițiat operația de transfer.

Limbajele de nivel înalt (**FORTRAN, PASCAL,** etc.) și standardul ISA nu permit legarea directă a task-urilor utilizator la mecanismul de validare/inhibare a întreruperilor **AST.**

6.4. Recuperarea din avarie

Recuperarea din avarie, implementată în cadrul **MIX**, permite sistemului să ignore fluctuațiile tensiunii de alimentare pe termen scurt, păstrînd integritatea sistemului (programe și date pe suporturi externe) și a task-urilor utilizator.

În momentul în care tensiunea de alimentare a sistemului de calcul scade sub limitele admise, acesta nu mai funcționează în mod corect. Dacă sistemul nu prevede un dispozitiv de recuperare din avarie (acumulatori

baterii), acesta se poate bloca datorită proastei funcționări a unora din echipamentele electronice. În schimb, sistemele prevăzute cu un astfel de dispozitiv, detectează fluctuațiile de tensiune generînd o întrerupere corespunzătoare către Monitor. Sistemul de recuperare hardware asigură apoi păstrarea tensiunii de alimentare un interval de timp suficient pentru ca Monitorul să salveze contextul curent înainte de căderea automată a sistemului.

De obicei, fluctuațiile tensiunii de alimentare durează cel mult cîteva secunde ; multe aplicații de timp real își pot permite o pierdere de informații în acest scurt interval de timp fără o degradare apreciabilă a performanțelor.

În sistemele care nu suportă recuperarea din avarie, întreruperile datorate căderii de tensiune au efect pe termen mai lung, ducînd la o reducere importantă a eficienței echipamentelor. Mai mult, redundanța sistemelor (un sistem de calcul suplimentar), nu îmbunătățește șansele în cazul unei fluctuații/căderi de tensiune. În cazul în care aplicația necesită păstrarea integrității sistemului, e necesară o sursă de alimentare suplimentară ce constituie, de obicei, o alternativă mult mai costisitoare.

Scopul recuperării din avarie este acela de restaurare logică totală a contextului întrerupt, cu minimum de pierderi. Aceasta implică, ca sistemul de operare să fie proiectat în așa fel încît să restaureze starea curentă anterioară căderii de tensiune, implicînd restaurarea conținutului memoriei și repoziționarea dispozitivelor de intrare/ieșire.

În cazul în care memoria este pe ferite, restaurarea memoriei se face simplu, aceasta nefiind afectată de pierderea de tensiune (cu excepția registrelor volatile din unitatea centrală, salvate în momentul căderii). În cazul în care memoria este pe semiconductori (volatilă), restaurarea acesteia se face prin păstrarea alimentării memoriei la un nivel scăzut (utilizînd acumulatori sau baterii).

Repoziționarea dispozitivelor de intrare/ieșire constituie o problemă mai complicată întrucît transferurile de intrare/ieșire în curs de realizare la momentul căderii de tensiune trebuie repetate. În plus, apare și întrebarea cît de transparentă este recuperarea din avarie pentru task-urile utilizator.

În sistemul de operare **MIX** se asigură o recuperare totală din avarie în *patru etape :*

a) În momentul în care tensiunea de alimentare scade sub limitele admise, Unitatea centrală generează o întrerupere de cădere de tensiune către Monitor care, după salvarea conținutului registrelor volatile în memoria nevolatilă, oprește funcționarea sistemului ;

b) În mementul în care tensiunea de alimentare revine în limitele admise, Unitatea centrală generează o întrerupere de refacere tensiune către Monitor, care restaurează starea anterioară căderii de tensiune ;

c) Monitorul planifică apoi în execuție toate driverele dispozitivelor de intrare/ieșire active în momentul căderii de tensiune, în punctele de intrare corespunzătoare tratării căderii de tensiune.

În funcție de dispozitivul periferic pe care îl servește, fiecare driver are opțiunea de a fi replanificat în execuție în unul din următoarele cazuri :

— la apariția căderii de tensiune, indiferent de starea dispozitivului de intrare/ieșire ;

— la apariția căderii de tensiune, numai dacă driver-ul prelucra o cerere de intrare/ieșire.

Fiecare driver poate, în felul acesta, să efectueze toate restaurările de stare necesare (de exemplu, repetarea unui transfer de intrare/ieșire) ;

d) Monitorul informează apoi, prin intermediul unor întreruperi software asincrone (**AST**), toate task-urile care au solicitat o tratare de acest tip (prin programarea unei directive Monitor **SPRA$**). Aceste task-uri utilizatorul pot, dacă este necesar, să restaureze acele stări (de exemplu, repetarea unor intrări/ieșiri) pe care le doresc. Mecanismul de întreruperi **AST** poate fi utilizat atît de task-urile privilegiate, cît și de cele neprivilegiate.

O astfel de tratare utilizator, în cazul sistemului de operare **MIX**, este eficientă, întrucît repetarea unor intrări/ieșiri este plasată cît mai aproape de sursa acestora — programul de aplicație.

7

Servicii și directive sistem MIX

7.1. Noțiuni privind serviciile sistem

Serviciile sistem sînt proceduri utilizate de către sistemul de operare **MIX** pentru controlul alocării resurselor disponibile task-urilor din sistem, pentru comunicarea și sincronizarea între task-uri și pentru realizarea unor funcții de bază ale sistemului de operare, cum ar fi de pildă coordonarea operațiilor de intrare/ieșire.

Deși majoritatea serviciilor sistem sînt utilizate de către sistemul de operare pentru servirea task-urilor utilizator, multe dintre acestea sînt de interes general și pot fi folosite în programele de aplicație. Anumite servicii trebuie însă utilizate cu grijă pentru a păstra performanțele sistemului și menține integritatea task-urilor utilizator.

Utilizatorii au acces la serviciile sistem prin intermediul *directivelor sistem*. O directivă sistem reprezintă o cerere a unui task utilizator către Monitor pentru executarea operației indicate.

Directivele sistem sînt utilizate în general pentru a controla execuția și interacțiunea dintre task-urile utilizator ; execuția anumitor directive duce la apariția unor evenimente semnificative. Evenimentele semnificative din sistem sînt datorate, direct sau indirect, execuției unor directive sistem. În felul acesta, directivele sistem constituie o parte integrantă a mecanismului de multiprogramare implementat în sistemul **MIX**, afectînd modul în care Monitorul distribuie resursele sistem între mai multe task-uri active simultan.

Directivele sistem permit task-urilor utilizator efectuarea următoarelor funcțiuni :

— obținerea unor informații sistem sau relative la task-uri ;
— comunicarea și sincronizarea între task-uri ;
— măsurarea intervalelor de timp ;
— replanificarea execuției altor task-uri ;
— creare și conectare de task-uri ;
— efectuarea unor operații de intrare/ieșire ;
— manipularea spațiilor de adresare virtual și logic, atașate task-urilor ;
— tratări de terminare, etc.

7.2. Implementarea directivelor sistem

Directivele sistemului **MIX** sînt implementate utilizînd instrucțiunea **EMT 377**. Codurile **EMT 0** la **EMT 376** (sau **375** pentru task-urile privilegiate, cu sau fără relocare) sînt considerate drept coduri nestandard **MIX** și utilizarea lor produce întreruperi software de tipul **SST**. Ca urmare a execuției unei asemenea directive, Monitorul abortează (termină anormal) execuția task-ului implicat, cu excepția cazului în care se specifică o tratare **SST** asociată instrucțiunii **EMT**. În cazul în care, în programele utilizator, se prevăd rutine de servire a întreruperilor software de tip **SST** pentru aceste coduri, se poate simula orice altă interfață sistem implementată de către utilizatori.

Pentru extensii ulterioare, sistemul de operare **MIX** rezervă codurile **EMT 370** la **EMT 376**.

Pentru a folosi directivele sistem, programatorii în limbajul **MACRO** utilizează macroinstrucțiuni sistem, conținute în biblioteca de macroinstrucțiuni sistem cu specificatorul : **LB : [1, 1] SYSMAC.SML.** Apelarea acestor macroinstrucțiuni se face prin specificarea explicită a numelor acestora ca argumente ale directivei de asamblare **.MCALL.**

Programatorii în limbaje de nivel înalt (**FORTRAN IV, FORTRAN-77, PASCAL, DIBOL-83, BASIC-PLUS-2, COBOL-81** etc.), utilizează directivele sistemului **MIX** prin apelarea unor subrutine conținute în biblioteca sistem cu specificatorul **LB : [1,1] SYSLIB.OLB.**

Întrucît aceste directive sînt tratate direct de către Monitor ele sînt cunoscute și sub numele de *directive Monitor.*

În tratarea unei directive sistem se disting *4 faze :*

— *Utilizatorul emite o directivă sistem.* Identificatorul directivei și parametrii asociați acesteia trebuie să fie plasați într-un bloc de parametri ai directivei (**DPB**), ce poate fi construit în stivă sau secțiunea de date a task-ului utilizator. Emiterea directivei se face prin programarea instrucțiunii **EMT 377** împreună cu adresa **DPB** sau chiar conținutul **DPB** în vîrful stivei task-ului apelant.

— *Monitorul primește o întrerupere software* (**SST**) corespunzătoare codului EMT și determină că este vorba de **EMT 377.**

— *Monitorul execută directiva sistem.* Primul cuvînt din fiecare **DPB** conține un cod de identificare al directivei (**DIC**) (octetul dreapta) și dimensiunea blocului **DPB** (octetul stînga). Codul de identificare indică tipul directivei sistem, iar dimensiunea este precizată în cuvinte.

— În final, *Monitorul returnează controlul task-ului utilizator,* cu o informație de stare a directivei.

Cu excepția anumitor directive, Monitorul returnează controlul la instrucțiunea următoare **EMT**, ștergînd sau poziționînd codul de condiție **C** din cuvîntul de stare program (**PS**) pentru a indica acceptarea (**C = 0**), respectiv, rejectarea (**C = 1**) directivei. Cauza specifică de acceptare/rejectare este precizată în cuvîntul de stare al directivei (**$DSW**).

La executarea unei directive Monitor, sistemul salvează și reface toate registrele task-ului apelant. După executarea directivei, adresa **DPB** sau conținutul **DPB**, plasat(ă) în stivă, este eliminat(ă). În cazul în care vîrful stivei conține referința (adresa) unui bloc **DPB**, după eliminarea acesteia

blocul **DPB** nu se modifică (putînd fi ulterior reutilizat). În cazul în care vîrful stivei conține chiar blocul **DPB**, eliminarea acestuia face imposibilă reutilizarea ulterioară. Departajarea între cele două situații de plasare în stivă este făcută de codul de identificare al directivei (**DIC**). Acest cod este întotdeauna impar ; de aceea Monitorul poate determina ușor dacă vîrful stivei conține adresa unui **DPB** (adresă pară) sau primul cuvînt din **DPB** (cuvînt impar).

7.3. Coduri de retur

După execuția unei directive, Monitorul returnează controlul task-ului apelant, la instrucțiunea următoare apelului de directivă (**EMT 377**), cu excepția directivelor **ASTX$S, EXIT$S, EXIF$, RCVX$** și parțial **RREF$** (în cazul specificării opțiunii de recepție condiționată).

Codul de retur rezultat în cuvîntul de stare al directivei (**$DSW**) poate aparține uneia din următoarele *clase* :

— cu o valoare pozitivă egală cu +1, în cazul acceptării sau executării cu succes a directivei (excepție făcînd codurile de retur valide pentru directivele **CLEF$, SETF$, GPRT$, GMCR$, CRVT$**) ;

— cu valori negative cuprinse între —1, și —21., semnificînd posibilitatea reluării execuției directivei după modificarea unor argumente sau luarea unor decizii utilizator (recuperare din eroare) ;

— cu valori negative cuprinse între —80. și —99., semnificînd apariția unor erori grave ce nu permit nici un fel de recuperare din eroare.

Codurile de retur pot fi testate simbolic prin utilizarea unei mnemonici ce reflectă cauza specifică de terminare, de forma

— **IS.***xyz*, pentru terminarea cu succes :

— **IE.***xyz*, pentru terminarea cu eroare (*xyz* reprezentînd forma abreviată a condiției de terminare).

7.4. Utilizarea directivelor sistem

Pentru a utiliza o directivă Monitor, un task specifică sistemului codul și parametrii directivei (în **DPB**) și execută instrucțiunea **EMT 377**.

Blocul de parametri ai directivei poate fi creat în două moduri :

a) *Dinamic, în stivă, la executarea task-ului.* Metoda necesită utilizarea formei **$S** a apelului de directivă, ce generează cod pentru introducerea **DPB**-ului în stivă, urmat de codul instrucțiunii **EMT 377**.

În programarea formei **$S** a apelului de directivă, parametrii specificați pentru construirea **DPB** trebuie să constituie operanzi sursă valizi pentru instrucțiunile de tip **MOV, MOVB** sau **CLR**. Parametrii de tip nume, formați din două cuvinte în format. **.RAD50**, se specifică prin adresa unui bloc de două cuvinte ce conține parametrul specificat. Parametrii de apel lipsă sînt înlocuiți cu parametri nul (instrucțiunea **CLR**).

Apel macro:

```
Nume$S #PAR1,#PAR2, ... #PARN [,#ERR]
```

Expandare:

```
MOV #PARN, -(SP)
      ⋮
MOV #PAR1,-(SP)
MOV (PC)+, -(SP)
.BYTE DIC, dimensiune
EMT 377
[BCC .+6
 JSR PC, ERR]
```

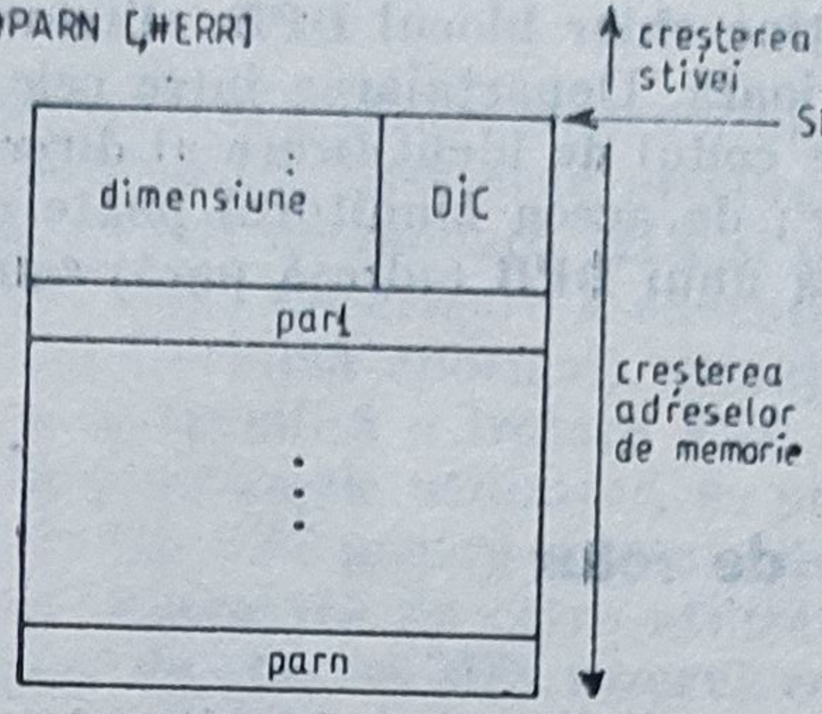


Fig. 7.1. Utilizarea formei $S a apelului de directivă

Forma $S acceptă de asemenea un argument opțional final ce specifică adresa unei rutine utilizator, ce urmează a fi apelată la apariția unor erori (C = 1) în execuția directivei Monitor.

Figura 7.1 ilustrează conținutul **DPB** în stivă și modul de utilizare a directivei Monitor.

Această formă este utilizată în general pentru scrierea programelor reentrante, avînd însă dezavantajul că blocul de parametri, generat în stivă, nu poate fi utilizat de către alte directive Monitor.

b) *Static, în momentul asamblării.*

În acest caz e necesară, mai întîi, crearea **DPB**-ului ca un bloc de date și apoi, generarea codului care specifică adresa **DPB** și execută instrucțiunea **EMT 377**. Aceasta se poate face în două feluri :

b.1. crearea **DPB** *și a codului executabil, cu macroinstrucțiuni diferite.* **DPB**-ul este creat utilizînd forma $ a apelului de directivă, după care programul poate apela directiva utilizînd macroinstrucțiunea **DIR$**, ce specifică adresa **DPB**-ului creat.

Codul **DPB** se generează întotdeauna în locul (secțiunea de program) în care se asamblează directiva și nu va include nici o instrucțiune executabilă. Parametrii de apel trebuie să constituie expresii valide pentru directivele de alocare a memoriei din limbajul MACRO (**.BYTE, .WORD, .RAD50**). Parametrii de apel lipsă sînt înlocuiți cu parametri de apel nuli (0).

Forma $ a apelului de directivă este recomandată în cazul în care programatorii doresc executarea aceleiași directive Monitor de mai multe ori. În acest caz, **DPB**-ul este generat o singură dată și toate apelurile se reduc la macroinstrucțiunea DIR$. În cazul în care parametrii de apel diferă, ei pot fi modificați dinamic, de către utilizator (prin specificarea unei etichete la începutul **DPB** și prin utilizarea unor deplasări simbolice, specifice fiecărei directive Monitor).

Figura 7.2 ilustrează conținutul **DPB** în stivă și modul de utilizare a directivei Monitor.

b.2. crearea DPB și a codului executabil cu o singură macroinstrucțiune. Acest lucru este posibil prin utilizarea formei $C a apelului de directivă, care generează **DPB**-ul și codul executabil în secțiuni de program diferite (separate fizic la editarea de legături).

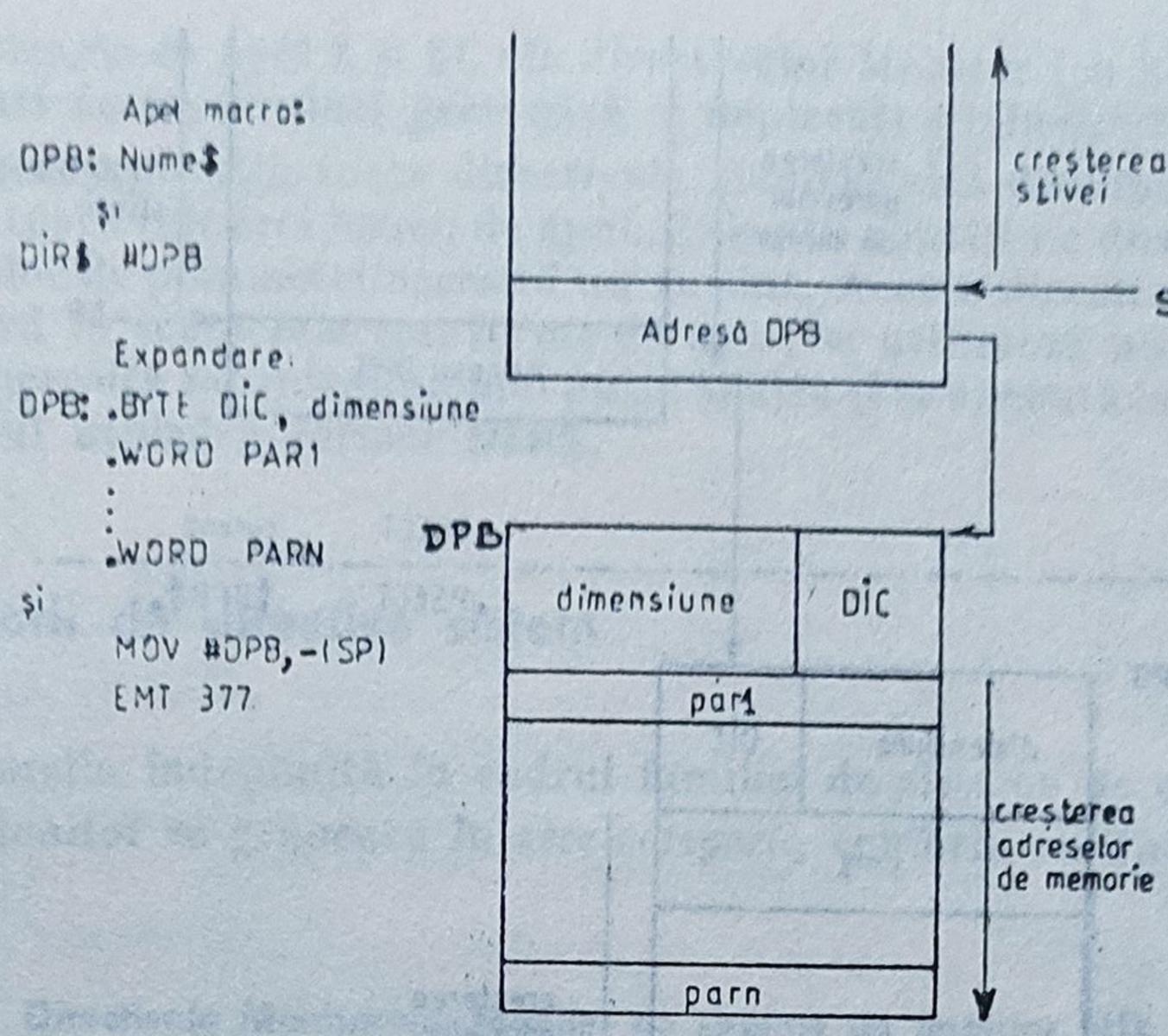


Fig. 7.2. Utilizarea formei $ a apelului de directivă

Codul **DPB** se generează întotdeauna într-o secțiune de program cu numele "**DPB.**". El este urmat de restabilirea numelui secțiunii de program inițiale (anterioare), de o instrucțiune (**MOV**) ce plasează adresa **DPB**-ului în stivă și de codul instrucțiunii **EMT 377.**

Pentru restaurarea corectă a secțiunii de program curente, este necesară precizarea, de către utilizator, a numelui de secțiune program în lista de argumente, după parametrii necesari construirii blocului **DPB** (dacă acest argument nu se specifică, programul continuă cu o secțiune de program fără nume — *blank* —). Forma $C acceptă de asemenea un argument opțional final ce specifică adresa unei rutine utilizator, ce urmează a fi apelată la apariția unei erori (C = 1) în execuția directivei Monitor.

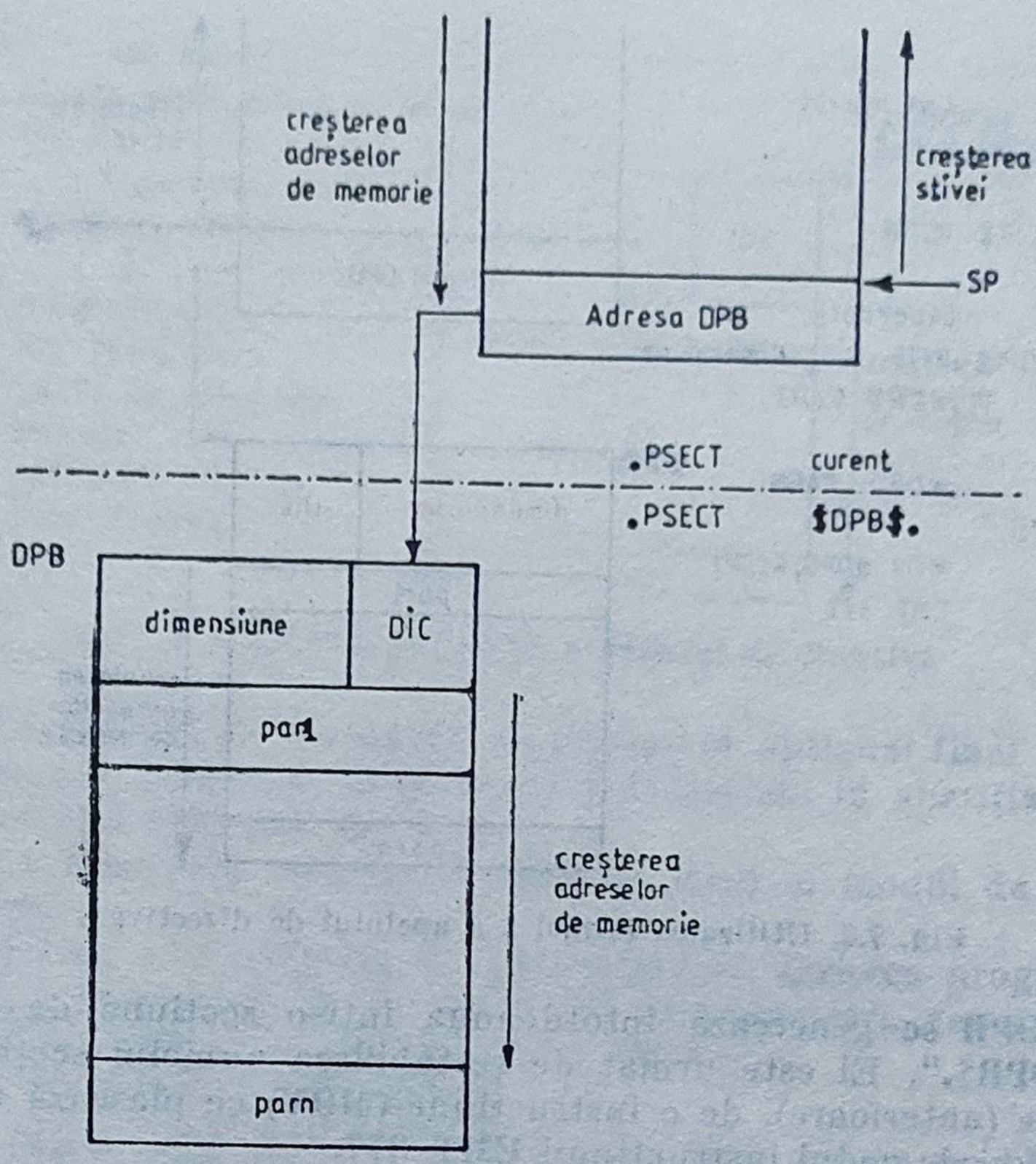
În programarea formei $C a apelului de directivă, parametrii specificați pentru construirea **DPB** trebuie să constituie expresii valide pentru directivele de alocare a memoriei din limbajul **MACRO (.BYTE, .WORD, .RAD50).** Fac excepție de la această regulă argumentul nume de secțiune program (ce trebuie să corespundă specificațiilor directivei de asamblare **.PSECT**) și argumentul adresă a unei rutine de eroare (ce trebuie să constituie un operand destinație valid pentru instrucțiunea **JSR**).

Parametrii de apel lipsă sînt înlocuiți cu parametri de apel nuli (0).

Utilizarea formei $C este recomandată pentru aplicațiile nereentrante, în care se utilizează **DPB**-ul creat o singură dată, constituind cea mai simplă formă de programare a directivelor Monitor. Ea are însă dezavantajul că **DPB**-ul astfel creat nu poate fi accesat din alte părți ale programului, întrucît adresa acestuia este necunoscută.

Eticheta asociată la asamblarea blocului **DPB** ($$$) va fi redefinită (și va avea alte valori), pentru fiecare nou apel de directivă cu formatul $C.

Figura 7.3 ilustrează modul de utilizare a formei $C a apelului de directivă.



Apel macro:

```
Nume$C     PAR1, ... PARN, psect curent [, ERR]
```

Expandare:

```
                .PSECT  $DPB$.
$$$ =.
                .BYTE   DIC, dimensiune
                .WORD   PAR1
                  :
                .WORD   PARN
                .PSECT  curent
                MOV     #$$$,-(SP)
                EMT     377
               [BCC     .+6
               [JSR     PC, ERR]
```

Fig. 7.3. Utilizarea formei $C a apelului de directivă

Majoritatea directivelor sistem generează *deplasări simbolice locale*, relative la blocul de parametri ai directivei sau diversele zone de lucru asociate. Întrucît aceste deplasări sînt definite simbolic, relativ la începutul **DPB**, programatorul nu are nevoie de o cunoaștere amănunțită a codului generat de fiecare macroinstrucțiune Monitor. Utilizarea deplasărilor simbolice elimină și rescrierea programelor în cazul schimbării organizării **DPB**, în cazul unor versiuni ulterioare ale sistemului de operare **MIX**.

Toate formele de apel **$** şi **$C** ale directivelor Monitor (cu blocuri de apel **DPB** mai mari de un cuvînt) generează şi deplasări simbolice locale.

Cu anumite excepţii, toate directivele Monitor sînt distribuite şi folosite standard, în toate cele trei forme de apel. Excepţii constituie directivele Monitor al căror bloc de parametri necesită un cuvînt. Aceste directive, deşi distribuite standard în toate cele trei forme de apel, se utilizează numai în forma **$S**, întrucît necesită întotdeauna mai puţin spaţiu şi se execută la fel de repede ca şi în cazul utilizării formei **DIR$**.

7.5. Categorii de directive sistem

Relativ la funcţia îndeplinită în cadrul familiei de sisteme de operare **MIX**, directivele Monitor se grupează în *zece categorii*, conform cu Tabela 7.1.

Tabela 7.1

Directivele Monitorului familiei de sisteme de operare MIX

Nr. crt.	DIRECTIVA	FUNCŢIA REALIZATĂ
0	1	2
		I. *DIRECTIVE DE CONTROL AL EXECUŢIEI TASK-URILOR*
1	**ABRT$**	Terminarea anormală a execuţiei task-ului
2	**CSRQ$**	Eliminarea sincronizărilor bazate pe timp
3	**EXIT$S**	Terminarea normală a execuţiei task-ului
4	**EXTK$**	Modificarea dimensiunii task-ului
5	**RQST$**	Cerere lansare în execuţie a task-ului
6	**RSUM$**	Relansarea în execuţie a task-ului
7	**RUN$**	Lansarea generală în execuţie a task-ului
8	**SPND$S**	Suspendarea execuţiei unui task
9	**SWST$**	Schimbarea stării curente a task-ului
		II. *DIRECTIVE DE CONTROL AL STĂRII TASK-URILOR*
10	**ALTP$**	Modificarea priorităţii de execuţie a task-ului
11	**DSCP$S**	Inhibarea evacuării task-ului din memoria internă
12	**ENCP$S**	Permiterea evacuării task-ului din memoria internă
		III. *DIRECTIVE DE INFORMARE*
13	**FEAT$**	Testarea unei caracteristici sistem specificate
14	**GPRT$**	Citirea parametrilor de descriere a partiţiei
15	**GREG$**	Citirea parametrilor de descrierea a regiunii
16	**GSSW$**	Citirea poziţiei comutatorilor de pe panoul de comandă
17	**GTIM$**	Citirea parametrilor de timp
18	**GTSK$**	Citirea parametrilor de descriere a task-ului
19	**TFEA$**	Testarea unei caracteristici task specificate
		IV. *DIRECTIVE ASOCIATE EVENIMENTELOR*
20	**CLEF$**	Ştergere indicator de eveniment
21	**CMKT$**	Eliminare sincronizări de timp, iniţiate prin directiva **MRKT$**
22	**CRGF$**	Creare grup de indicatori globali de evenimente

Tabela 7.1 (continuare)

0	1	2
23	DECL$S	Declarare eveniment semnificativ
24	ELGF$	Eliminare grup de indicatori globali de evenimente
25	EXIF$	Terminare condiţionată a execuţiei task-ului
26	MRKT$	Sincronizare task, bazată pe timp
27	RDAF$	Citire indicatori de evenimente
28	RDXF$	Citire extinsă a indicatorilor de evenimente
29	SETF$	Poziţionare indicator de eveniment
30	STIM$	Stabilire timp sistem
31	STLO$	Stopare multiplă pe indicatori de evenimente
32	STOP$S	Stopare execuţie task
33	STSE$	Stopare pe indicator de eveniment
34	ULGF$S	Deblocare acces la grup de indicatori globali de evenimente
35	USTP$	Eliminarea stopării execuţiei unui task
36	WSIG$S	Aşteptare apariţie eveniment semnificativ
37	WTLO$	Aşteptare multiplă pe indicatori de evenimente
38	WTSE$	Aşteptare pe indicator de eveniment
		V. *DIRECTIVE ASOCIATE ÎNTRERUPERILOR SOFTWARE*
39	ASTX$S	Terminare tratare AST
40	DSAR$S	Inhibare generală recunoaştere AST
41	ENAR$S	Permitere generală recunoaştere AST
42	IHAR$S	Inhibare generală recunoaştere AST
43	SCAA$	Permitere tratare AST recepţie comenzi
44	SFPA$	Permitere tratare AST virgulă mobilă
45	SPEA$	Permitere tratare AST eroare de paritate
46	SPRA$	Permitere tratare AST cădere de tensiune
47	SRDA$	Permitere tratare AST recepţie mesaje
48	SREAR$	Permitere tratare AST terminare execuţie task
49	SREX$	Permitere tratare AST terminare condiţionată execuţie task
50	SRRA$	Permitere tratare AST recepţie referinţă
51	SVDB$	Validare vector SST pentru sistemele de depanare
52	SVTK$	Validare vector SST pentru task
		VI. *DIRECTIVE DE INTRARE/IEŞIRE ŞI COMUNICARE ÎNTRE TASK-URI*
53	ALUN$	Asignare număr logic
54	CINT$	Conectare/deconectare la/de la o întrerupere
55	GLUN$	Citire informaţii ataşate unui număr logic
56	GMCR$	Preluare linie de comandă MCL
57	QIO$	Cerere de intrare/ieşire
58	QIOW$	Cerere de intrare/ieşire cu aşteptare
59	RCST$	Recepţie date sau stopare
60	RCVD$	Recepţie date
61	RCVX$	Recepţie date sau terminare execuţie
62	SDAT$	Emisie date
63	SMSG$	Emisie mesaj către subsistemul de înregistrare erori
64	VRCD$	Recepţie date de lungime variabilă
65	VRCS$	Recepţie date de lungime variabilă sau stopare
66	VRCX$	Recepţie date de lungime variabilă sau terminare execuţie
67	VSDA$	Emisie date de lungime variabilă
		VII. *DIRECTIVE DE GESTIONARE A MEMORIEI*
68	ARTG$	Ataşare la regiune
69	CRAW$	Creare fereastră de adrese (segment virtual)
70	CRRG$	Creare regiune dinamică

Tabela 7.1 (continuare)

0	1	2
71	**DTRG$**	Detașare de la regiune
72	**ELAW$**	Eliminare fereastră de adrese (segment virtual)
73	**GMCX$**	Citire context de mapare
74	**MAP$**	Mapare fereastră de adrese (segment virtual
75	**RREF$**	Recepție referință
76	**RRST$**	Recepție referință sau stopare
77	**SREF$**	Emisie referință
78	**UMAP$**	Ștergerea proiecției unei ferestre de adrese (segment virtual)
	VIII. *DIRECTIVE CE DESCRIU RELAȚIILE DE PATERNITATE ÎNTRE TASK-URI*	
79	**CNCT$**	Conectare la task
80	**EMST$**	Emitere stare
81	**EXST$**	Terminare normală execuție task cu emitere stare
82	**RPOI$**	Cerere lansare în execuție task cu transmitere informații de paternitate
83	**SDRC$**	Emisie date, cerere lansare în execuție task și conectare
84	**SDRP$**	Emisie date, cerere lansare în execuție task și transmitere informații de paternitate
85	**SPWN$**	Lansare generală în execuție task (creare subtask)
86	**VSRC$**	Emisie date de lungime variabilă, cerere lansare în execuție task și conectare
	IX. *DIRECTIVE PENTRU INTERFAȚAREA INTERPRETOARELOR DE LIMBAJE DE COMANDĂ* **(CLI)**	
87	**GCCI$**	Obținere linie de comandă asociată unui task de tip **CLI**
88	**GCII$**	Obținere informații asociate unui task de tip **CLI**
89	**SCLI$**	Asociere de tip **CLI** unui terminal
	X. *DIRECTIVE SPECIFICE SISTEMULUI DE OPERARE* **MIX-PLUS**	
90	**ACHN$**	Asignare canal
91	**CLON$/CLOG$**	Creare nume logic
92	**CPCR$**	Evacuare regiune de comun
93	**CRVT$**	Creare terminal virtual
94	**DLON$/DLOG$**	Eliminare (ștergere) nume logic
95	**ELVT$**	Eliminare (desființare) terminal virtual
96	**FSS$**	Analizare specificator de fișier
97	**GDIR$**	Citire catalog implicit
98	**GIN$**	Citire informații specifice sistem
99	**MSDS$**	Mapare spațiu **D** atașat modului *Supervizor*
100	**MVTS$**	Citire/scriere din/în spațiu **I** sau **D**
101	**PFCS$**	Parsare specificator de fișier și construire bloc de descriere **FCS**
102	**PRMS$**	Parsare specificator de fișier și construire bloc de descriere **RMS**
103	**RDEF$**	Citire indicator de eveniment singular
104	**RLON$/RLOG$**	Translatare recursivă (iterativă) a unui nume logic
105	**RMAF$S**	Eliminare afinități
106	**SCAL$S**	Apelarea modulului *Supervizor*
107	**SDIR$**	Stabilire catalog implicit
108	**SNXC$**	Transmiterea comenzii următoare
109	**STAF$**	Stabilire afinități

Lista apelurilor de subrutine din limbajele de nivel înalt, asociate directivelor Monitorului **MIX**, este prezentată în tabela 7.2.

Tabela 7.2

Lista apelurilor de subrutine din limbajele de nivel înalt asociate directivelor Monitorului MIX

Directivă	Subrutină
ABRT$	**CALL ABORT**
ACHN$	**CALL ACHN**
ALTP$	**CALL ALTPRI**
ALUN$	**CALL ASNLUN**
ASTX$S	Indisponibilă
ATRG$	**CALL ATRG**
CINT$	Indisponibilă
CLEF$	**CALL CLREF**
CLON$	**CALL CRELON**
CLOG$	**CALL CRELOG**
CMKT$	**CALL CANMT**
CNCT$	**CALL CNCT** **CALL CNCTN**
CPCR$	**CALL CPCR**
CRAW$	**CALL GRAW**
CRGF$	**CALL CRGF**
CRRG$	**CALL CRRG**
CRVT$	**CALL CRVT**
CSRQ$	**CALL CANALL**
DECL$S	**CALL DECLAR**
DLON$	**CALL DELLON**
DLOG$	**CALL DELLOG**
DSAR$S	**CALL DSASTR**
DSCP$S	**CALL DISCKP**
DTRG$	**CALL DTRG**
ELAW$	**CALL ELAW**
ELGF$	**CALL ELGF**
ELVT$	**CALL ELVT**
EMST$	**CALL EMST**
ENAR$S	**CALL ENASTR**
ENCP$S	**CALL ENACKP**
EXIF$	**CALL EXITIF**
EXIT$S	**CALL EXIT**
EXST$	**CALL EXST**
EXTK$	**CALL EXTTSK**
FEAT$	**CALL FEAT**
FSS$	**CALL FSSFSS**
GCCI$	**CALL GTCMCI**
GCII$	**CALL GETCII**
GDIR$	**CALL GETDDS**
GIN$	Indisponibilă
GLUN$	**CALL GETLUN**
GMCR$	**CALL GETMCR**
GMCX$	**CALL GMCX**
GPRT$	**CALL GETPAR**
GREG$	**CALL GETREG**
GSSW$S	**CALL READSW** **CALL SSWTCH**
GTIM$	**CALL GETTIM**
GTSK$	**CALL GETTSK**

Tabela 7.2 (continuare)

Directivă	Subrutină
IHAR$S	**CALL INASTR**
MAP$	**CALL MAP**
MRKT$	**CALL MARK** **CALL WAIT (standard ISA)**
MSDS$	Indisponibilă
MVTS$	Indisponibilă
PFCS$	**CALL PRSFCS**
PRMS$	**CALL PRSRMS**
QIO$	**CALL QIO**
QIOW$	**CALL WTQIO**
RCST$	**CALL RCST**
RCVD$	**CALL RECEIV**
RCVX$	**CALL RECOEX**
RDAF$	**CALL READEF** (poate citi numai un indicator de eveniment)
RDEF$	**CALL READEF** (poate citi numai un indicator de eveniment)
RDXF$	**CALL READEF** (poate citi numai un indicator de eveniment)
RLON$	**CALL RCTLON**
RLOG$	**CALL RCTLOG**
RMAF$S	**CALL RMAF**
RPOI$	**CALL RPOI**
RQST$	**CALL REQUES**
RREF$	**CALL RREF**
RRST$	**CALL RRST**
RSUM$	**CALL RESUME**
RUN$	**CALL RUN** **CALL START (standard ISA)**
SCAA$	Indisponibilă
SCAL$S	Indisponibilă
SCLI$	**CALL SCLI**
SDAT$	**CALL SEND**
SDIR$	**CALL SETDDS**
SDRC$	**CALL SDRC** **CALL SDRCN**
SDRP$	**CALL SDRP**
SETF$	**CALL SETEF**
SFPA$	Indisponibilă
SMSG$	**CALL SMSG**
SNXC$	**CALL SNXC**
SPEA$	Indisponibilă
SPND$S	**CALL SUSPND**
SPRA$	**EXTERNAL SUBNAM** **CALL PWRUP (SUBNAM)** — pentru stabilire AST **CALL PWRUP** — pentru eliminare AST
SPWN$	**CALL SPAWN** **CALL SPAWNN**
SRDA$	Indisponibilă
SREA$	**CALL SREA**
SREX$	**CALL SREX**
SREF$	**CALL SREF**
SRRA$	Indisponibilă

Tabela 7.2 (continuare)

Directivă	Subrutină
STAF$	**CALL STAF**
STIM$	**CALL SETTIM**
STLO$	**CALL STLOR** / **CALL STLORS**
STOP$S	**CALL STOP**
STSE$	**CALL STOPFR**
SVDB$	Indisponibilă
SVTK$	Indisponibilă
SWST$	Indisponibilă
TFEA$	**CALL TFEA**
TLON$	**CALL TRALON**
TLOG$	**CALL TRALOG**
ULGF$S	**CALL ULGF**
UMAP$	**CALL UNMAP**
USTP$	**CALL USTP**
VRCD$	**CALL VRCD**
VRCS$	**CALL VRCS**
VRCX$	**CALL VRCX**
VSDA$	**CALL VSDA**
VSRC$	**CALL VSRC** / **CALL VSRCN**
WSIG$S	**CALL WFSNE**
WTLO$	**CALL WFLOR** / **CALL WFLORS**
WTSE$	**CALL WAITFR**

8

Interfețe operator — sistem MIX

8.1. Controlul sistemului de către operator

Exceptînd aplicațiile de timp real, dedicate, fără consolă operator, sistemul de operare **MIX** necesită prezența constantă a unui *operator (utilizator)*, centralizînd funcțiile de inițiere, lansare în execuție și control pe unul sau mai multe dispozitive de intrare/ieșire (terminale).

Termenul de operator se referă la orice persoană care intră în contact cu sistemul de operare. Pentru configurațiile de calcul mici (prelucrări de date de laborator, instalații în timp real, conducere și control de procese etc.). La un utilizator poate supraveghea funcționarea sistemului de calcul fără a-și neglija activitățile sale curente. În cazul unei configurații de lucru cu multe terminale, discuri și benzi magnetice, pe care se execută operații de gestiune mică-medie, e necesară prezența unei persoane specializate (operatorul), pentru a supraveghea funcționarea echipamentelor și a gestiona programele de aplicație introduse.

Utilizatorii comunică cu sistemul de operare **MIX** prin intermediul unor comenzi speciale, introduse de la unul sau mai multe terminale.

Tratarea comenzilor operator este efectuată de un task sistem numit *Procesor al comenzilor operator*, care execută singur anumite comenzi sau activează un număr de task-uri sistem sau utilizator *(funcții operator extinse)* pentru prelucrarea acestora.

Interfața operator-sistem permite operatorului :

— inițializarea și lansarea sistemului ;
— gestiunea echipamentelor hardware ;
— controlul execuției task-urilor ;
— obținerea unor informații sistem sau utilizator ;
— activarea unor task-uri sistem sau utilizator ce solicită intrări/ieșiri de la terminalul(ele) operator.

8.2. Cerințe privind interfața operator-sistem

În proiectarea interfeței operator — sistem **MIX** au fost luate în considerare următoarele aspecte :

— drepturi de acces ;
— limbajul de comunicare ;

— adaptabilitatea la configurații dinamice ;
— răspuns optim la cerințele de timp real.

Drepturile de acces se referă la rolul operatorului în cadrul sistemului de operare și, de egală importanță, la funcția unei console operator (terminal).

În aplicații ce implică o deplină securitate a datelor și a programelor (cu acces multiplu, de la distanță), accesul operatorilor trebuie să fie limitat la informații generale privind starea sistemului și la informații particulare privind starea aplicației corespunzătoare.

Limbajul de comunicare necesită un astfel de format, pe cît de simplu, pe atît de eficient în exploatarea posibilităților de lucru ale sistemului. Conținutul limbajului este determinat de drepturile de acces ale operatorului la un moment dat.

Adaptabilitatea la configurații dinamice. Interfața operator a sistemului de operare **MIX** este proiectată în scopul modificării și extinderii ușoare a acesteia, din nevoia adaptării interfeței de comunicare operator-sistem la diversitatea aplicațiilor utilizator.

Răspuns optim la cerințele de timp real. Interfața operator a sistemului a fost proiectată în așa fel încît aceasta să nu monopolizeze serviciile sistem în detrimentul aplicațiilor de timp real.

Pornind de la aceste aspecte, interfața operator-sistem **MIX** asigură :
— acces deplin al operatorului la sistem ;
— posibilitatea unor intervenții directe, de urgență, pentru corectarea anomaliilor software ;
— simplicitate de utilizare și de modificare sau îmbunătățire.

8.3. Configurații de lucru operator

În funcție de specificul aplicației conduse de sistemul de operare **MIX**, pot exista următoarele configurații de lucru operator :
— fără terminal de comandă ;
— monoterminal ;
— multiterminal.

Configurații fără terminal de comandă. Acestea pot apare în cazul unor aplicații specializate, incluse în cadrul unui proces sau unei instalații complexe, în care nu se poate plasa local un terminal de comandă sau în care controlul operatorului nu este necesar.

În cazul în care această configurație cuprinde unul sau mai multe terminale, ele sînt utilizate pentru extragerea centralizată a datelor aplicației sau introducerea locală a unor date necesare task-urilor utilizator.

Configurații monoterminal. Aceste configurații sînt frecvente, în aplicațiile mici și medii. Operatorul are un acces deplin la resursele sistem, putînd controla direct sau lua decizii de urgență asupra aplicației curente.

Configurațiile monoterminal sînt recomandate aplicațiilor de timp real (de orice complexitate) și aplicațiilor mici de dezvoltare de programe (memoria disponibilă sub 32 kcuvinte).

Configurații multiterminal. Aceste configurații se întîlnesc în cazul unor aplicații combinate, de timp real și dezvoltare de programe sau în cazul unor

aplicații de time-sharing (tranzacționale, exploatare bază de date, „front-end“ în aplicații de interconectare, etc.).

Operatorii acestei configurații au un acces diferențiat la resursele sistemului de calcul *(utilizatori privilegiați* și *neprivilegiați)*, putînd controla de obicei aplicațiile corespunzătoare fiecăruia. În aceste configurații, sistemul de operare **MIX** implementează un mecanism software de protecție între utilizatori, pentru asigurarea integrității sistemului și performanțelor generale de exploatare ale acestuia.

8.4. Caracteristicile unui terminal

În sistemul de operare **MIX**, pot exista și funcționa în paralel un număr variabil de terminale, fiecare lucrînd independent de celelalte din sistem.

Caracteristicile unui terminal depind de tipul și modul de utilizare al acestuia. În sistemul de operare **MIX** se definesc următoarele caracteristici de exploatare ale unui terminal :

— privilegii ;
— atașare/detașare ;
— aservire ;
— coduri de identificare utilizator.

Privilegiile terminalelor sînt determinare *static*, la generarea driver-ului de terminal sau *dinamic* de către operatorii privilegiați și condiționează dreptul operatorilor de a executa anumite comenzi sau opțiuni de comenzi.

Corespunzător acestor privilegii, operatorii se împart în :

— operatori privilegiați ;
— operatori neprivilegiați.

În configurațiile multiterminal, existența mai multor operatori privilegiați trebuie considerată cu grijă, întrucît aceștia pot interfera în mod destructiv cu sistemul de operare și între ei.

În configurațiile multiterminal fără protecție, utilizatorul poate emite comenzi către orice task, de la orice terminal al configurației. Starea (privilegiată sau neprivilegiată) a terminalului este fixată la generarea bazei de date a driver-ului de terminal și poate fi schimbată, dinamic, prin comenzi operator. În configurațiile multiterminal cu protecție, utilizatorul trebuie să deschidă o sesiune de lucru, înaintea introducerii oricărei comenzi dorite (prin comanda operator **MCL HELlO** sau **DCL LOGin**). La introducerea în sistem, terminalul capătă privilegiile utilizatorului, în funcție de valoarea codului său de identificare (**UIC** — User Identification Code), alocat de inginerul de sistem într-un fișier de conturi. **UIC**-urile cu cod de grup mai mic sau egal cu 10(8) sînt privilegiate ; restul de **UIC**-uri sînt neprivilegiate.

Schimbarea privilegiilor unor operatori (terminale) este permisă numai operatorilor privilegiați, prin utilizarea comenzii operator **SET** (opțiunea **PRIV** sau **NOPRIV**).

Atașare/Neatașare. Un terminal este *atașat* atunci cînd este utilizat de către un alt task decît Procesorul de comenzi operator (**MCL, DCL** sau **CLI**).

Pe un astfel de terminal nu se pot efectua transferuri de intrare/ieşire decît la iniţiativa task-ului care a solicitat ataşarea. De exemplu, la introducerea unui program sursă de la terminal, constituind intrarea pentru Editorul de texte **MIX**, acesta îşi ataşază terminalul de intrare pentru ca nici un alt utilizator să nu poată efectua transferuri de intrare/ieşire pe el. La terminarea introducerii programului, Editorul de texte detaşează terminalul de intrare, următoarele introduceri de date (comenzi) fiind dirijate, din acest moment, către Procesorul de comenzi operator (**MCL, DCL** sau **CLI**).

În cazul unui terminal ataşat, atenţionarea Procesorului de comenzi operator se face prin introducerea unui caracter de control special (**CTRL/C**).

Terminalele care nu sînt utilizate exclusiv de alte task-uri, ci şi de Procesorul de comenzi operator, se numesc *terminale neataşate*.

Ataşarea/detaşarea unui terminal sînt funcţii ale task-urilor şi nu ale interfeţei operator-sistem.

Aservire. Un terminal capătă statutul de *aservit* atunci cînd este dedicat utilizării exclusive de către unul sau mai multe task-uri din sistem.

Obţinerea sau desfiinţarea statutului de aservit e posibilă prin executarea unei comenzi operator **SET** (opţiunea **SLAVE**, respectiv **NOSLAVE**), emisă de la un terminal privilegiat, de către un task printr-o funcţie de intrare/ieşire specială, sau la deschiderea unei sesiuni de lucru.

Diferenţa între un terminal ataşat şi unul aservit este aceea că de la un terminal aservit nu se poate atenţiona Procesorul de comenzi operator şi nu se acceptă intrări nesolicitate, cu excepţia caracterelor de control **CTRL/O, CTRL/Q** şi **CTRL/S**. Un terminal ataşat acceptă unele forme de intrări nesolicitate, cum ar fi **CTRL/C**, caractere speciale sau caractere de control al formatului de tipărire.

Pînă la desfiinţarea statutului de aservit, un terminal poate comunica numai cu task-ul ce îl utilizează în mod exclusiv. Aceste tipuri de terminal sînt dedicate, de obicei, aplicaţiilor de timp real, ce necesită un control continuu şi neîntreruptibil al transferurilor de intrare/ieşire.

Coduri de identificare utilizator (**UIC**)

În cadrul sistemului, fiecare terminal posedă un cod de identificare utilizator (**UIC**), stabilit static la generarea driver-ului de terminal, sau modificat dinamic, prin comanda operator (**SET**, cu opţiunea **UIC**). Acest cod determină drepturile de acces ale unui utilizator la un fişier, atunci cînd acesta exploatează structura de fişiere **MIX**.

În sistemele **MIX**, implementate cu opţiunea (standard) de protecţie între utilizatori, fiecare terminal are asociate două **UIC**-uri: unul pentru *protecţie* şi altul *implicit*. **UIC**-ul de protecţie determină drepturile de acces ale unui utilizator la un fişier. **UIC**-ul implicit este utilizat pentru identificarea unui fişier în cazul în care se omite cîmpul de **UIC** din specificatorul de fişier, determinînd catalogul (**UFD**) în care urmează a fi căutat acest fişier.

Modulele de acces la fişiere (**FCS** şi **RMS**) verifică **UIC**-ul de protecţie cu masca de protecţie a fişierului pentru a determina drepturile de acces ale utilizatorului la acel fişier.

Modul în care sistemul de operare setează cele două **UIC**-uri depinde de starea utilizatorului, după introducerea sa în sistem. Pentru un utilizator neprivilegiat, **UIC**-ul de protecţie este identic cu cel de introducere în sistem; **UIC**-ul implicit este de asemenea egal cu cel de introducere în sistem. O co-

mandă operator ulterioară **SET/UIC** va permite modificarea **UIC**-ului implicit ; în acest caz, **UIC**-ul de protecţie rămîne neschimbat.

La introducerea în sistem a unui utilizator privilegiat, **UIC**-ul de protecţie şi **UIC**-ul implicit devin egale cu **UIC**-ul de introducere în sistem. O comandă operator ulterioară **SET/UIC** va permite modificarea ambelor **UIC**-uri (de protecţie şi implicit).

În felul acesta se interzice unui utilizator neprivilegiat alterarea **UIC**-ului de protecţie ataşat terminalului.

8.5. Caractere speciale şi de control

Operatorul controlează lucrul pe terminal prin intermediul unor *caractere speciale şi de control.*

Caracterele speciale sînt :

— retur de car (**CR**) = terminator al liniei de intrare ;

— tabulare orizontală (**TAB**) sau **CTRL/I**) = separator în linia de intrare ;

— ştergere caracter (**RUBOUT** sau **DELETE**) = şterge ultimul caracter introdus de la terminal şi pe cele anterioare, dacă este introdus repetat ;

— **ESCAPE** (sau **ALTMODE**) = terminator al liniei de intrare cu semnificaţie specială, etc.

Caracterele de control sînt produse prin apăsarea simultană a două clape (**CTRL** — control şi un alt caracter).

Caracterele de control importante sînt :

— **CTRL** şi **C** = atenţionează Procesorul de comenzi operator (**MCL**, **DCL** sau **CLI**) că se doreşte introducerea unei comenzi. Sistemul **MIX** emite identificatorul Procesorului de comenzi operator (**MCL >**, **DCL >** sau cel asociat **CLI**) şi solicită introducerea unei comenzi. Aceasta constituie singura formă de atenţionare a Procesorului de comenzi de la pupitrul unui terminal ataşat ;

— **CTRL** şi **K** = salt vertical (avans) patru linii ;

— **CTRL** şi **L** = avans opt linii (similar caracterului **FF**, fără paginare) ;

— **CTRL** şi **O** = suprimarea şi reluarea alternativă a afişării (extragerii de date) pe terminal ;

— **CTRL** şi **Q** = reluare ieşire suspendată anterior de **CTRL/S** ;

— **CTRL** şi **R** = retipărire linie curentă cu suprimarea caracterelor şterse (driverul de terminal comandă afişarea caracterelor ↑ **R**) ;

— **CTRL** şi **S** = suspendarea ieşirii pe terminal pînă la introducerea **CTRL/Q** sau **CTRL/C**. Ieşirea suspendată nu se pierde ca în cazul **CTRL/O** ;

— **CTRL** şi **U** = suprimarea introducerii liniei curente. Sistemul simulează introducerea **CTRL/U** la depăşirea valorii de time-out între două caractere consecutive (uzual — 2 minute). În ambele cazuri driverul de terminal comandă afişarea caracterelor ↑ U).

— **CTRL** şi **X** = ştergerea buferului de acumulare („typeahead") ataşat terminalului ;

— **CTRL** şi **Z** = specificator de sfîrşit de fişier, utilizat pentru semnalizarea terminării introducerii de date de la terminal în cazul componentelor

MIX de dezvoltare de programe. Driverul de terminal comandă afișarea caracterelor ↑Z.

— **CTRL** și **Y** = forțează terminarea anormală a task-ului ce tipărea (afișa) informații pe terminal.

8.6. Introduceri de date solicitate și nesolicitate

Starea normală a unui terminal operator este de așteptare a introducerii unor comenzi. Starea este identificată de caracterul „>" în prima poziție a unei noi linii de intrare. Termenul introducere nesolicitată implică faptul că nu există în sistem un task care să fi solicitat obținerea unor date (comenzi) de la terminal.

În acest caz, orice introducere a unei linii de intrare este automat transmisă Procesorului de comenzi operator, care o recepționează, identifică, analizează și răspunde corespunzător (introducere eronată sau comandă operator). Caracterul „>" este numit și *identificator implicit al Procesorului de comenzi operator* sau *„prompter"*.

În cazul unor terminale atașate, introducerile de date sînt transmise task-urilor care au solicitat acest lucru și nu Procesorului de comenzi operator. Obținerea unor date (linii de comandă) solicitate, se face prin scrierea, pe terminalul operator atașat, a unei secvențe de forma *xyz* > (unde *xyz* reprezintă un identificator al numelui de task format din trei caractere), urmată de așteptarea unei introduceri de date.

Acționarea forțată a Procesorului de comenzi operator e posibilă, indiferent de starea terminalului operator (cu excepția terminalelor aservite) prin introducerea caracterelor de control **CTRL** și **C**. În acest caz, Procesorul de comenzi operator răspunde prin identificatorul explicit **MCL** >, **DCL** > sau cel asociat unui **CLI** specific, urmat de așteptarea introducerii unei linii de comandă. După prelucrarea liniei de comandă introduse, se revine la starea anterioară cu excepția cazului în care prelucrarea comenzii a modificat starea terminalului de intrare.

8.7. Facilități de acces operator

Facilitățile de acces operator permit unui utilizator interactiv, aflat la pupitrul unui terminal, să interacționeze direct cu sistemul de operare, prin intermediul unor comenzi operator generale sau specifice. Aceste facilități includ :

— *Limbajul standard de comandă* al familiei de sisteme de operare **MIX** — **MCL** (**MIX** *Command Language*), ce constituie interfața de bază între utilizatorii interactivi și sistemul de operare) ;

— *Limbajul extins de comandă* — **DCL** *(DIGITAL Command Language)*, bazat pe o sintaxă engleză, constituind interfața între utilizatorii interactivi și alte componente ale sistemului de operare ;

— *Interpretoarele de limbaje de comandă* — **CLI** (Command Language Interpreters), constituind interfețe specifice între utilizatorii interactivi și sistemul de operare sau aplicațiile utilizator ;

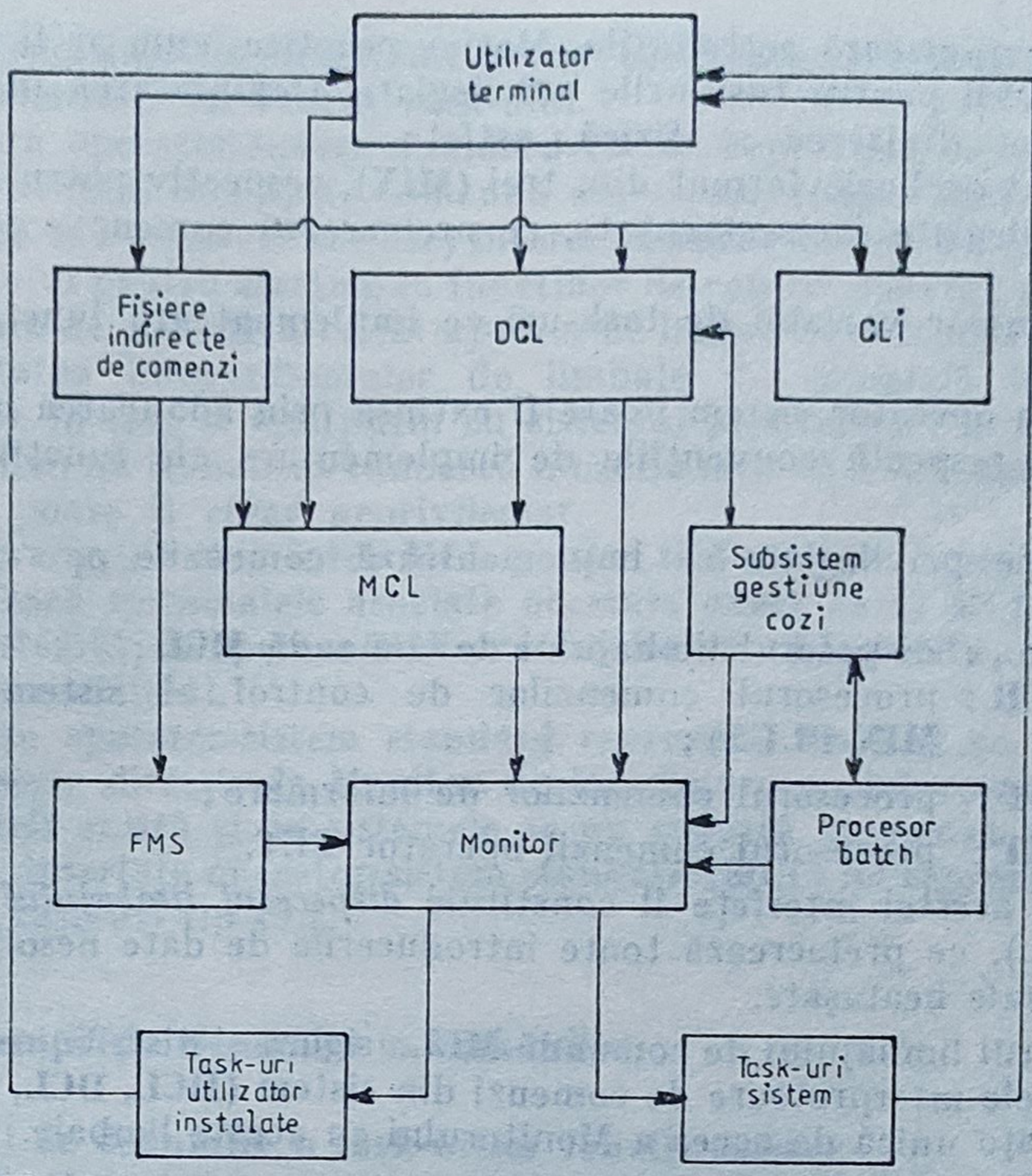


Fig. 8.1. Facilităţile de acces operator în sistemele de operare MIX/MIX-PLUS

— *Fişierele indirecte de comenzi*, ce permit utilizatorilor interactivi să definească şi să memoreze secvenţe des utilizate de comenzi operator (**MCL, DCL** sau **CLI**), pe care le pot folosi ulterior, în mod repetitiv, prin precizarea numelui fişierului ce conţine aceste comenzi ;

— *Subsistemul de gestiune a cozilor* — **QMG** (Queue Manager), ce asigură facilităţi de tipărire asincronă cu cozi multiple de intrare/ieşire *(spooling)* şi prelucrări de tip *batch* ;

— *Subsistemul de gestiune a videoformatelor* — **FMS** (Forms Management System), ce asigură un control interactiv al formatelor pe dispozitive de afişare tip display.

Fig. 8.1 reprezintă facilităţile de acces operator existente în sistemele de operare **MIX/MIX—PLUS** şi relaţiile între grupele de componente sistem implicate.

8.8. Interfaţa operator sistem standard (MCL)

8.8.1. Implementarea interfeţei operator-sistem

Cunoscută sub numele de *limbaj standard de comandă* al familiei de sisteme de operare **MIX—MCL (MIX Command Language)** — interfaţa operator-sistem este implementată sub forma unor task-uri sistem sau utilizator care comu-

nică și își sincronizează activitățile. Motive practice (cum ar fi mărimea codului executabil pentru task-urile privilegiate, flexibilitatea în configurare au determinat divizarea sa fizică, astfel :

— un set de bază, format din, trei (**MIX**), respectiv patru (**MIX-PLUS**) task-uri privilegiate (ne)segmentate, ce prelucrează comenzile operator propriu-zise ;

— un număr variabil de task-uri ce implementează funcțiile operator extinse.

Interfața operator sistem poate fi extinsă prin adăugarea unor task-uri utilizator ce respectă convențiile de implementare ale funcțiilor operator extinse.

Task-urile privilegiate ce implementează comenzile operator propriu-zise sînt :

— **MCR...** : dispecerul limbajului de comandă **MCL** ;
— **...MCR** : procesorul comenzilor de control al sistemului (numai **MIX-PLUS**) ;
— **...LST** : procesorul comenzilor de informare ;
— **...SET** : procesorul comenzii operator **SET**.

Nucleul acestei interfețe îl constituie *dispecerul limbajului de comandă* **MCL (MCR...)**, ce prelucrează toate introducerile de date nesolicitate emise de la terminale neatașate.

Dispecerul limbajului de comandă **MCL** asigură și distribuirea comenzilor între limbajele interpretoare de comenzi din sistem (**MCL, DCL, CLI**), constituind interfața unică de acces a Monitorului cu aceste limbaje.

Dispecerul **MCL** administrează și comenzile operator proprii prin distribuirea controlului către task-urile din setul de bază sau către funcțiile operator extinse.

În sistemul de operare **MIX**, cu excepția comenzilor de informare și **SET**, majoritatea comenzilor operator propriu-zise (în special cele care nu interacționează cu structura de fișiere **MIX** sau nu au un volum mare de intrări/ieșiri) sînt prelucrate tot de dispecerul **MCL** (task-ul **MCR...**), în ordinea în care sînt introduse de la terminale.

În sistemul de operare **MIX-PLUS**, comenzile operator propriu-zise (cu excepția comenzilor de informare și **SET**) sînt prelucrate de procesorul comenzilor de control al sistemului (**...MCR**), dispecerul **MCL (MCR...)** păstrînd numai funcțiile de distribuire a comenzilor între diversele limbaje interpretoare de comenzi din sistem. Aceasta permite o servire și o prelucrare rapidă a liniilor de comandă, generalizînd interfața de acces sistem (Monitor) — limbaje de comandă.

Un alt grup de comenzi, numite *funcții operator extinse*, sînt implementate sub forma unor task-uri sistem sau utilizator. Aceste task-uri sînt identificate printr-un nume de forma ***...tsk***, unde tsk reprezintă forma compactă a comenzii (3 caractere).

În sistemul de operare **MIX**, funcții operator extinse sînt comenzile notate cu (E) în listele din subcapitolul 8.8.7.

Alegerea unei astfel de soluții oferă o mare flexibilitate în implementarea unor noi comenzi operator de către utilizatori. Aceștia își pot implementa funcții utile aplicației curente sub forma unor task-uri pe care le pot apela

prin simpla specificare a unui nume format din 3 caractere, urmat de eventualii parametri analizați în cadrul task-ului.

Interfața operator-sistem standard (**MCL**) reprezintă cel mai important interpretor al unui limbaj de comandă din sistem (**CLI**). Task-urile sale (din setul de bază sau funcțiile extinse) interacționează extrem de puternic cu Nucleul sistemului pentru efectuarea funcțiilor de control sistem ; această funcție nu este legată de rolul **MCL** ca interpretor de limbaj de comandă (**CLI**).

Majoritatea interpretoarelor de limbaje de comandă interacționează cu aplicațiile în sine și mai puțin cu sistemul propriu-zis ; proiectantul unui astfel de limbaj nu trebuie să cunoască filozofia internă a sistemului, iar task-ul interpretor poate fi chiar neprivilegiat.

Singura similitudine între **MCL** și alte **CLI**-uri este aceea că orice task **CLI** controlează terminalele asociate acestuia exact ca și **MCL**. Fiecare terminal în sistemul de operare **MIX** are asociat un **CLI** specific, ce controlează utilizarea terminalului.

Interfața operator-sistem standard reprezintă un **CLI** cu totul special. Ea nu folosește directivele Monitor de interfațare cu interpretoarele de comenzi întrucît există și pe sistemele ce nu suportă asemenea directive. Din acest motiv, interfața operator-sistem standard (**MCL**) nu reprezintă un model pentru alte task-uri tip **CLI**.

8.8.2. Convenții de denumire a task-urilor

a) **Convenții de denumire a task-urilor funcții extinse**

Funcțiile operator extinse, precum și alte componente ale familiei de sisteme de operare **MIX** (utilitare, compilatoare, subsisteme, etc.), sînt implementate sub forma unor task-uri cu numele ...*tsk*.

În sistemele fără protecție între utilizatori, o singură copie a unui astfel de task poate fi activă la un moment dat (task-ul ...*tsk*), ceea ce necesită coordonarea acțiunilor utilizatorilor aflați la diverse terminale.

b) **În sistemele cu protecție între utilizatori**

Sistemele generate cu protecție între utilizatori (**MIX/MIX-PLUS**) permit utilizatorilor aflați la diferite terminale să poată lucra independent unul față de celălalt. Aceasta se realizează prin crearea cîte unei copii per terminal pentru același task. La prima cerere de execuție a unei funcții se lansează în execuție task-ul ...*tsk*. Dacă înainte de terminarea task-ului ...*tsk*, sistemul primește de la un alt terminal o altă cerere de execuție a aceleiași funcții, se creează o copie temporară a task-ului primar.

Numele noului task este *tsk***T***nn* sau *tsk***V***nn*, unde :

tsk — reprezintă numele de 3 litere al comenzii ;

T — comanda a fost introdusă de la un terminal fizic ;

V — comanda vine de la un terminal virtual ;

nn — numărul terminalului de la care s-a introdus comanda.

Pentru unitățile 0 ÷ 77, *nn* corespunde numărului octal al unității. Pentru unitățile 100 ÷ 377, sistemul definește *nn* utilizînd următorul algoritm :

Unitate	100	101	...107	110...117	120...370...377
nn	A0	A1	...A7	B0...B7	C0...X0...X7

De exemplu, dacă de la terminalul **TT2 :** s-a cerut execuția unei funcții a task-ului utilitar **PIP** în timp ce task-ul **...PIP** era deja activ, se creează o copie numită **PIPT2.** Orice comandă ulterioară referitoare la acest task va trebui să specifice pe poziția „*numetask*" :

.PIP sau **PIPT2,** dacă comanda se introduce de la terminalul **TT2** ;

.PIPT2, dacă comanda se introduce de la orice alt terminal.

După terminarea execuției sale, copia temporar creată (task-ul **PIPT2** din exemplul de mai sus) este automat eliminată din sistem. După terminarea execuției task-ului ...*tsk*, acesta continuă să rămînă instalat în sistem.

În sistemul de operare **MIX-PLUS**, un task cu numele ...*tsk* se numește *task prototip* și nu este niciodată lansat în execuție. La activarea task-ului cu numele „*tsk*", se crează întotdeauna o copie temporară a task-ului prototip, numele și evoluția acestei copii respectînd convențiile descrise anterior.

c) În sistemele cu copii multiple per terminal

În mod obișnuit, de la un terminal poate fi activată cel mult o copie a unui task funcție extinsă. Sistemul de operare **MIX** (NU și **MIX-PLUS**) suportă opțiunea de *copii multiple per terminal* ale aceluiași task funcție extinsă ; în acest fel, utilizatorul poate solicita sistemului executarea unei noi funcții înainte ca task-ul ...*tsk* ce o prelucrează, și pe care tot el l-a lansat în execuție, să-și fi treminat execuția. Deci, spre deosebire de sistemele cu protecție între utilizatori în care caracteristica este lansarea cîte unei copii per terminal, aici este vorba de crearea mai multor copii de task pentru unul și același terminal.

La prima cerere de activare, sistemul lansează în execuție task-ul ...*tsk*, unde task reprezintă numele de 3 litere al comenzii. Dacă înainte de terminarea execuției task-ului ...*tsk*, utilizatorul de la același terminal solicită din nou execuția funcției, sistemul creează o *copie temporară* cu numele *tsknn***A** unde :

tsk — reprezintă numele de 3 litere al comenzii ;
nn — numărul unității terminalului ;
A — identificatorul de copie.

O nouă cerere de execuție va determina sistemul să creeze copia *tsknn***B.** Ultima copie ce se poate crea este *tsknn***Z.** Pentru ca utilizatorul să aibă controlul asupra proliferării acestor copii, s-a adoptat mecanismul numerotării lor strict secvențiale. Astfel, dacă copia **D** este activă, **A, B** și **C** terminîndu-și execuția, sistemul va crea copia *tsknn***E.**

Orice comenzi ulterioare referitoare la vreuna din copiile *tsknn***A**, *tsknn***B** ... *tsknn***Z** trebuie să specifice numele complet al task-ului, adică *tsknn***A**, *tsknn***B**, ... *tsknn***Z.**

După terminarea execuției lor, copiile sînt automat eliminate din sistem.

Facilitatea de creare de copii multiple per terminal este independentă de crearea copiilor în regim de protecție între utilizatori și se poate utiliza dacă :

- sistemul a fost generat cu opțiunea de suport pentru task-uri nerezidente ;
- sistemul a fost generat cu opțiunea de copii multiple per terminal ;
- task-ul ...*tsk* a fost instalat într-o partiție controlată de sistem.

d) **Task-uri cu instalare, execuţie, eliminare**

Pentru a putea executa un task oarecare, procedura de urmat este următoarea :

— se instalează task-ul (comanda **INS**) ;

— se cere lansarea lui în execuţie (comanda **RUN**).

După terminarea execuţiei task-ul continuă să rămînă în sistem.

Interfaţa operator-sistem standard (**MCL**) pune la dispoziţia utilizatorilor un mod mai simplu de a efectua cele de mai sus. Aceasta se realizează prin intermediul celei de-a *cincea forme a comenzii* **RUN**. Printr-o singură linie de comandă (corespunzătoare lui **RUN**) se specifică sistemului toţi parametrii necesari pentru a face *instalarea*, *execuţia* şi *eliminarea* din sistem a task-ului indicat. Pe durata cît task-ul este activ el poartă numele **TT***nn*, unde *nn* este numărul unităţii terminalului.

Atîta timp cît task-ul este activ, el poate fi referit cu numele :

— **TT***nn*, indiferent de terminalul de la care se introduce comanda respectivă ;

— fără a specifica nici un nume, dacă comanda este introdusă de la terminalul **TT***nn* : de la care a fost lansat în execuţie şi task-ul **TT***nn*.

8.8.3. Utilizarea MCL prin relaţii de paternitate

În mod obişnuit, comenzile către Procesorul de comenzi operator (prin aceasta înţelegîndu-se atît task-ul **MCR...** cît şi funcţiile operator extinse) sînt introduse de la terminal. Există însă posibilitatea ca un task utilizator sau sistem să transmită o comandă către **MCL** prin intermediul directivelor Monitor **SPWN$** sau **RPOI$**. Opţional, la programarea acestor directive se poate specifica o linie de comandă şi un terminal de intrare (**TI :**) asociat unei unităţi de terminal virtual sau fizic. Task-ul poate fi scris în orice limbaj de programare suportat de sistem.

În urma transmiterii unei comenzi prin directiva **SPWN$** sau **RPOI$**, între task-ul emiţător şi Procesorul de comenzi operator se creează relaţii de parternitate, dintre care primul este task-ul „tată“, iar cel de-al doilea — task-ul „fiu“.

La sfîrşitul execuţiei comenzii, task-ul „fiu“ comunică task-ului „tată“ codul de terminare a execuţiei comenzii. În principiu, acesta poate fi :

EX$SUC — cod de succes ;

EX$ERR — cod de eroare.

După cum s-a amintit, Procesorul de comenzi operator este divizat fizic în mai multe task-uri apelabile de către dispecerul interfeţei operator-sistem (task-ul **MCR...**). În cazul cînd execuţia unei comenzi reclamă apelarea unui astfel de task, acesta devine task „fiu“ în locul lui **MCR....** El comunică în final codul de terminare a execuţiei comenzii.

Utilizarea Procesorului de comenzi operator prin relaţii de paternitate este folosită şi de task-ul „*catch-all*“, furnizat standard în sistemele **MIX** / **MIX-PLUS**.

8.8.4. Facilitatea de „catch-all"

O facilitate importantă pusă la dispoziția utilizatorilor este aceea de *„catch-all"* („agață tot"). La introducerea unei comenzi, dacă aceasta nu face parte din cele prelucrate de dispecerul interfeței operator (task-ul **MCR...**), dispecerul trimite comanda task-ului cu numele ...*tsk*, ceea ce presupune că task-ul în cauză trebuie să fie instalat. Dacă nu este instalat și la generare s-a selectat opțiunea *„catch-all"*, comanda este trimisă unui task cu numele ...**CA**. (dacă este instalat). Dacă nici acest task nu este în sistem sau nu există opțiunea *„catch-all"*, se emite un mesaj de eroare.

Task-ul *„catch-all"*, furnizat odată cu sistemul de operare, implementează un mecanism care asigură utilizarea mai eficientă a unei memorii dinamice și crearea unui nou limbaj de comandă.

Utilizarea puțin frecventă a multora din task-urile sistem folosite de utilizator face ca prezența lor permanentă în sistem să nu fie justificată. O cale de remediere a acestei situații este instalarea lor de către utilizator ori de cîte ori este nevoie și apoi eliminarea lor din sistem prin comanda **REM.** Acest mod de lucru ar îngreuna considerabil procedura de operare de la terminal. Scopul primar al facilității de „catch-all" este, prin urmare, acela de a permite utilizatorilor execuția de task-uri neinstalate.

Scopul secundar al facilității de *„catch-all"* este acela de a accepta un număr de pseudo-comenzi, pe care le translatează în comenzi **MCL** și le lansează în execuție :

Pseudo-comandă	**Translatarea**	**Semnificația**
ATS	**ACT/ALL**	Afișare listă totală task-uri active ;
ATS [TT*nn* :]	**ACT/TERM [= TT*nn* :]**	Afișare listă task-uri active, lansate de la terminalul specificat ;
CHD	**SET/DEF**	Afișare director curent ;
CHD *g, m*	**SET/DEF** = [*g, m*]	Schimbare director curent ;
CHU	**SET/UIC**	Afișare **UIC** curent ;
CHU *g, m*	**SET/UIC** = [*g, m*]	Schimbare **UIC** curent cu *g, m* ;
CLR		Ștergere ecran display ;
CRE *fișier*	**PIP** *fișier* = **TI :**	Creare fișier nou ;
CVT expresie numerică		Comandă generală de conversie ;
DEL [*fișier(e)*]	**PIP** *fișier(e)*]/**DE**	Ștergere fișier(e) ;
DIR [*fișier(e)*]	**PIP TI :** = [*fișier(e)*]/**LI**	Listare catalog la terminalul emițător ;
DLG	**DEV/LOG**	Afișare listă utilizatori introduși în sistem ;
DLN	**NCP SHO KNOWN NODES**	Afișare nume noduri cunoscute, conectate la rețeaua **MININET/MIX** ;
FRE [*ddnn*] :	**PIP** [*ddnn :*]/**FR**	Afișare număr blocuri libere pe volum/dispozitiv ;

PUR [*fișier(e)*]	**PIP** [*fișier(e)*]/**PU**	Ștergere fișier(e)';
SHQ	**QMG SHOW QUEUE ALL**	Afișare conținut cozi de intrare/ieșire ;
IDN sau		Listare identificator pentru
TDX		task-ul **CATCH** ;
SYS	**SET/SYSUIC**	Afirare **UIC** sistem ;
TYP [*fișier(e)*]	**PIP TI : =** [*fișier(e)*]	Listare fișer(e) pe terminal.

Dacă comanda introdusă nu seamănă cu una de mai sus, task-ul **CATCH** va permite următoarea comandă către dispecerul interfeței operator :

RUN $XXX/TASK = XXXTNN/CMD = "XXX [*param*]"

Caracterele **XXX** reprezintă numele task-ului și „*param*" reprezintă linia de comandă introdusă de utilizator. Dacă fișierul task este găsit în **UIC**-ul sistem, va fi instalat și lansat în execuție cu transmiterea liniei de comandă specificată. La terminarea execuției, se comandă eliminarea acestuia din sistem.

Un utilizator are posibilitatea să-și creeze un task propriu „*catch-all*", cu care să înlocuiască task-ul sistem standard ale cărui funcțiuni au fost descrise mai sus.

8.8.5. **Sintaxa comenzilor operator**

O comandă operator este introdusă ca răspuns la *identificatorii impliciți* (**>**) sau *expliciți* (**MCL >**) ai Procesorului de comenzi operator. Componentele unei linii de comandă sînt :

— numele comenzii, sub o formă compactă (3 litere) sau extinsă ;

— parametrii asociați comenzii (de obicei un nume de task, un specificator de fișier, un nume de dispozitiv de intrare/ieșire) urmați de comutatori și cuvinte cheie ;

— un terminator de linie (caracterul retur de car (**CR**) sau **ALTMODE** (**ESCAPE**)).

Lungimea unei linii de comandă **MCL** este de max. 80 (10) caractere (inclusiv terminatorul de linie) pentru sistemul de operare **MIX** și de max 255 (10) caractere pentru sistemul de operare „**MIX-PLUS**" (care admite și linii continuare, specificate prin caracterul „—" pe post de caracter terminator).

8.8.6. **Mesaje de eroare operator**

În familia de sisteme de operare **MIX**, pentru o identificare rapidă a comenzii pentru care s-a semnalizat eroare, mesajele de eroare operator sînt prefixate de forma compactă a numelui comenzii.

În sistemul de operare **MIX-RT** se poate selecta o formă scurtă a mesajelor de eroare (numele comenzii urmat de un număr de eroare), utilă în sistemele cu configurații mici de memorie, sau o formă extinsă (numele comenzii urmat de un mesaj complet de explicitare a erorii), recomandată pentru configurațiile medii-mari de memorie, facilitate standard în sistemele de operare **MIX/MIX-PLUS**.

În cazul formei extinse a mesajelor de eroare, la generarea sistemului se poate selecta una din următoarele două limbi de comunicație operator —

sistem : română, engleză (toate sistemele de operare **MIX** sînt distribuite în mod standard cu mesaje de comunicare în limba engleză). Extinderea setului de mesaje de eroare poate fi făcută de utilizator în cazul implementării unor noi comenzi operator.

8.8.7. Categorii de comenzi operator

Interfața operator-sistem standard (**MCL**) este asigurată de un număr de comenzi, ce pot fi împărțite în cinci categorii, descrise mai jos.

Pentru a limita utilizarea unor comenzi ce afectează performanțele generale ale sistemului, Procesorul de comenzi operator consideră anumite comenzi sau opțiuni ale acestora drept privilegiate, permițînd introducerea lor numai de la terminalele privilegiate (specificația **P** la descrierea comenzilor). Comenzile implementate ca funcții operator extinse sînt marcate cu specificația **E.**

Comenzi de inițializare

ACD		(P)	Încărcare/descărcare rutine de traducere a setului de caractere per terminal ;
ACS	(E)	(P)	Alocare/dezalocare fișier (spațiu) de evacuare pe un volum disc
ASN		(P)	Definire sau eliminare asociere dintre un dispozitiv logic și unul fizic. Listare asocieri curente pe terminalul de intrare ;
BOO[T]	(E)	(P)	Încărcare sistem în memorie și transferare control către acesta ;
DFL		(P)	Definire, vizualizare sau eliminare asocieri (atribuiri) nume logice. Listare asocieri curente pe terminalul de intrare ;
DMO[UNT]	(E)	(P)	Invalidare recunoaștere *(demontare)* volum logic (comandă complementară **MOUNT**) ;
FLA[GS]		(P)	Creare, vizualizare sau eliminare indicatori de evenimente globali, de grup ;
HOM[E]	(E)	(P)	Modificare informații în blocul de descriere *(home)* al unui volum disc cu structură **FILES-11** ;
INI[TVOLUME]	(E)	(P)	Inițializare volum magnetic în format **MIX** (**FILES-11** pentru volume disc, **ANSI** pentru volume bandă) ;
INS[TALL]	(E)	(P)	Instalare task în sistem (cu sau fără fixare în memorie) ;
LOA[D]	(E)	(P)	Încărcare în memorie driver nerezident (încărcabil) sau extensie Monitor vectorizată ;
MOU[NT]	(E)	(P)	Validare recunoaștere *(montare)* volum logic ;
SET		(P)	Stabilire sau modificare caracteristici sistem, caracteristici ale terminalelor, (re)definire partiții, vizualizare informații sistem, stabilire coduri de identificare utilizator, etc. ;

TIM[E]		(P)	Inițializare/vizualizare timp și dată curentă ;
UFD	(E)	(P)	Creare fișier de directori **(UFD)**, în fișierul principal de directori **(MFD)** de pe un volum disc ;
UNL[OAD]	(E)	(P)	Eliminare din sistem driver nerezident sau extensie Monitor vectorizată (comandă complementară **LOAD**).

Comenzi de informare asupra stării sistemului

ACT	Vizualizare nume task-uri active ;
ATL	Vizualizare nume task-uri active, precum și alte informații legate de acestea ;
CBD	Vizualizare nume și stare pentru regiunile de comun și bibliotecile de rutine, rezidente, instalate în sistem ;
CLQ[UEUE]	Vizualizare informații despre task-urile ce au solicitat sincronizări de timp ;
DEV[ICES]	Vizualizare informații despre dispozitivele fizice din sistem ;
LUN[S]	Vizualizarea asocierilor curente dintre numerele logice **(LUN)** și dispozitivele de I/E pentru un task ;
PAR[TITIONS]	Vizualizare informații de stare referitoare la partițiile din sistem ;
TAL	Vizualizare informații de stare relative la toate task-urile din sistem ;
TAS[KLIST]	Vizualizarea informații succinte de stare relative la toate task-urile instalate în sistem.

Comenzi de control al execuției task-urilor

ABO[RT]	(P)	Terminare execuție task activ ;
ALT[ER]	(P)	Modificare prioritate task ;
BLK	(P)	Blocare execuție task activ ;
CAN[CEL]	(P)	Anulare sincronizări bazate pe timp atașate unui task ;
FIX	(P)	Fixare task nerezident în memorie ;
REA[SSIGN]	(P)	Modificarea asocierii număr logic **(LUN)** — dispozitiv de intrare/ieșire pentru un task ;
RED[IRECT]	(P)	Redirectarea cererilor de I/E adresate unui dispozitiv fizic de I/E către un altul ;
REM[OVE]	(P)	Eliminare task/zonă partajată din sistem (comandă complementară **INSTALL**) ;
RES[UME]	(P)	Reluare execuție task suspendat ;
RUN	(P)	Planificare externă execuție task ;

Aceasta constituie o funcție importantă a interfeței operator-sistem, acționînd în conjuncție cu planificarea internă, bazată pe

priorităţi, a task-urilor active. Parametrii de timp specificaţi în comanda **RUN**, permit Monitorului **MIX** lansarea în execuţie a unui task : la un moment de timp specificat faţă de momentul curent ; la un moment de timp specificat faţă de unitatea de sincronizare a ceasului sistem ; la un moment de timp absolut ; imediat.
Toate aceste opţiuni sînt disponibile cu sau fără o planificare periodică ;

UNB[LOCK]	(P)	Deblocare task-anterior blocat (comandă complementară **BLK**) ;
UNF[IX]	(P)	Defixare task din memorie (comandă complementară **FIX**) ;
UNS[TOP]	(P)	Reluare execuţie task anterior stopat de către Monitor.

Comenzi de întreţinere şi depanare sistem

BRK	(P)	Apelarea subsistemului de depanare Monitor (**MDS**) ;
DEB[UG]	(P)	Apelarea subsistemului de depanare utilizator (**DEBUG**) (numai sub **MIX-PLUS**) ;
OPE[N]	(P)	Vizualizarea sau modificarea conţinutului unei locaţii de memorie ;
SAV[E]	(E) (P)	Salvarea unei imagini sistem (memorie) pe un dispozitiv extern ;
SSM	(P)	Inserare mesaj în fişierul de înregistrare a erorilor ;
SWR	(P)	Setare, ştergere sau vizualizare bit în registrul de comutatori ai Unităţii Centrale (numai pe sistemele **MIX-PLUS** multiprocesor).

Comenzi pentru asigurarea protecţiei între utilizatori

ALL[OCATE]	(P)	Alocare dispozitiv pentru utilizare particulară ;
BYE	(E)	Închidere sesiune de lucru utilizator ;
BRO[ADCAST]	(E)	Emitere mesaj către unul sau mai multe terminale ;
DEA[LLOCATE]	(P)	Dealocarea unui dispozitiv particular (comandă complementară **ALLOCATE**) ;
HEL[LO] sau **LOG[IN]**	(E)	Deschidere sesiune de lucru utilizator ;
HELP	(E)	Afişare informaţii ajutătoare.

8.9. Interfața operator-sistem extinsă (DCL)

8.9.1. Generalități

O caracteristică importantă a sistemelor de operare **MIX/MIX-PLUS** o constituie extensia interfeței operator-sistem, printr-un limbaj suplimentar de comandă, numit **DCL** (**DIGITAL Command Language**), ce facilitează accesul unui utilizator de limbă engleză la sistem, în mod interactiv.

Limbajul **DCL** este orientat către operații de tipul **COPY** sau **LINK**, față de operațiile similare **MCL** efectuate de task-urile **PIP** sau **TKB**. Interfața **DCL** cu utilizatorul unui terminal este complet diferită de cea specifică **MCL**, fiind mai ușor de învățat și utilizat. Utilizînd comenzile **DCL**, utilizatorul unui terminal nu trebuie să știe nimic despre alte limbaje de comandă, task-uri utilitare etc.

Interfața operator-sistem extinsă, **DCL**, este implementată sub forma unui task sistem (**DCL...**), numit *Procesor de comenzi* **DCL**, ce constituie un interpretor de comenzi de tip **CLI**. Procesorul **DCL** translatează de obicei sintaxa comenzilor **DCL** în comenzi **MCL** echivalente.

Utilizatorii limbajului **DCL** pot emite următoarele tipuri de comenzi:

— comenzi specifice **DCL**;

— comenzi **MCL**, echivalente unor comenzi **DCL**;

— comenzi adresate unor task-uri sistem sau utilizator, echivalente unor comenzi **DCL**;

Precizarea tipului de limbaj de comandă atașat unui terminal se poate face *static*, la crearea numerelor de cont atașate unor utilizatori interactivi (precizîndu-se denumirea **CLI**-ului atașat utilizatorului) sau *dinamic*, prin comandă operator (**SET/DCL = TI:**).

8.9.2. Caracteristici de operare

La introducerea unei comenzi **DCL** de la un terminal, Procesorul de comenzi **DCL** citește comanda, verifică sintaxa comenzii și determină tipul de prelucrare cerut prin comandă. În cazul unei comenzi **DCL**, aceasta este prelucrată direct de Procesorul **DCL**. Dacă comanda solicită activarea altei componente a sistemului de operare, Procesorul **DCL** lansează în execuție task-ul necesar și pasează controlul acestui task.

La completarea prelucrării, controlul revine Procesorului **DCL** care va accepta intrări suplimentare de la terminalul utilizator.

Prompter-ul implicit al limbajului **DCL** (emis la introducerea **CTRL/C**) este **DCL >**.

Procesorul de comenzi **DCL** include un algoritm propriu de tratare a liniilor de continuare, în cazul comenzilor introduse pe mai multe linii.

8.9.3. Categorii de comenzi DCL

Există șapte categorii de comenzi **DCL** standard descrise în subcapitolele care urmează.

Fiecare comandă **DCL** admite unul sau mai mulți *calificatori generali* sau *specifici*.

Comenzi de Dezvoltare de Programe

Aceste comenzi sînt translatate de Procesorul **DCL** în apeluri ale unor task-uri de dezvoltare de programe.

BASIC Apelare interpretor **BASIC** sau compilator **BASIC-PLUS-2**;

COBOL Apelare compilator **COBOL** sau **COBOL-81**, pentru obținere modul obiect, din unul sau mai multe fișiere sursă și execuție acțiuni specificate de unul sau mai mulți calificatori;

DEBUG Apel subsistem de depanare utilizator (numai pentru task-uri legate prin comanda **LINK/DEBUG**);

EDIT Apelare editor de texte specificat:

/EDI (Implicit) pentru editorul de texte mod linie;
/EDT pentru editorul de texte mod ecran;
/SOS pentru editorul SOS, nesuportat;
/SLP apelează Utilitarul de corecție programe;
/TECO, /MAKE sau **/MUNG** pentru editorul și corectorul de Texte **(TECO)**, nesuportat;
/USING: identificator-utilizator pentru un editor utilizator.

FORTRAN Apelare compilator **FORTRAN-IV, FORTRAN-IV-PLUS** sau **FORTRAN-77**, pentru obținere modul obiect din unul sau mai multe fișiere sursă și execuție acțiuni specificate de unul sau mai mulți calificatori;

LIBRARY Apelare bibliotecar **(LBR)** pentru execuție funcții specifice;

LINK Apelare Editor de legături **(TKB)** pentru obținere imagine task;

MACRO Apelare Macroasamblor **MACRO** pentru obținere modul obiect relocatabil;

SET GROUPFLAGS Creare/ștergere indicatori de evenimente globali de grup;

SHOW GROUPFLAGS Afișare indicatori de evenimente globali de grup existenți în sistem.

Comenzi de manipulare și control fișiere

Comenzile din această categorie permit utilizatorului să apeleze task-uri sistem de manipulare și tipărire de fișiere.

APPEND Apelare **PIP**, pentru adăugarea conținutului unuia sau mai multor fișiere la fișierul de ieșire;

ARCHIVE Apelare **BRU** pentru operațiile de punere la zi și restaurare;

CONVERT Apelare **RMSCNV** pentru conversia înregistrărilor de un tip, dintr-un fișier, în alt tip, în alt fișier;

COPY Apelare **PIP** pentru copiere fișier de intrare în fișier de ieșire;

CREATE Creare fișier secvențial;

CREATE/DIRECTORY Creare catalog pe volum cu structură **FILES-11**;

DELETE Apelare **PIP** pentru ștergere fișier(e) specificat(e);

DIFFERENCES Comparare două fișiere de text (ASCII), linie cu linie;

DIRECTORY Afișare informații despre fișierele aparținînd unui catalog;

MERGE Apelare **RMSCNV** pentru inserare înregistrări într-un fișier relativ sau indexat, creere fișier secvențial, sau adăugare înregistrări la un fișier secvențial existent;

PRINT Apelare subsistem de gestiune cozi — **QMG** — pentru introducerea job-ului specificat într-o coadă de ieșire a procesorului de tipărire, pentru listare ulterioară ;

PURGE Apelare **PIP** pentru ștergerea tuturor versiunilor unui fișier în afară de ultima versiune ;

RENAME Apelare **PIP** pentru schimbarea numelui, tipului sau numărului de versiune al(e) fișierului(elor) de intrare ;

SORT Apelare **SORT** pentru citire fișier, sortare conținut fișier și scriere date sortate într-un fișier de ieșire ;

SUBMIT Apelare subsistem de gestiune cozi — **QMG** — pentru introducerea job-ului specificat în coada procesorului batch (de prelucrare în loturi), pentru execuție ulterioară ;

TYPE Apelare **PIP** pentru tipărire fișiere selectate pe terminalul utilizatorului ;

UNLOCK Deblocare fișier blocat.

Comenzi de administrare și întreținere sistem

Aceste comenzi permit inginerului de sistem să administreze caracteristicile resurselor sistem.

SET BUFFER-POOL Setarea dimensiunii zonei cu alocare dinamică Monitor (**DSR**), la valoarea specificată ;

SET [DAY] TIME Stabilire timp și/sau dată curentă ;

SET DEFAULT Asignarea numelui logic **SY:** la dispozitivul specificat sau apelare **MCL** pentru execuția comenzii **SET/UIC** ;

SET DEVICE Setarea dispozitivului specificat cu caracteristicile specificate ;

SET EXTENSION LIMIT Setarea limitei maxime pînă la care un task se poate extinde ;

SET LIBRARY UFD Setarea catalogului de biblioteci sistem (**UFD**) la **UIC**-ul specificat ;

SET LIBRARY/DIRECTORY (MIX-PLUS) Stabilire catalog sistem (**UFD**) pentru utilitare sistem și task-uri sistem neprivilegiate ;

SET LOGINS Permite introducerea în sistem a utilizatorilor ;

SET NOLOGINS Interzicerea introducerii în sistem a utilizatorilor ;

SET NOPARTITION Eliminare partiție specificată ;

SET PACKETS Setare număr de pachete I/E prealocate ;

SET PARTITION Setare partiție specificată ;

SET PRIORITY Schimbare prioritate pentru task-ul activ specificat ;

SET PROTECTION Setare cod de protecție pentru fișierul specificat ;

SET QUEUE Setare una sau mai multe opțiuni pentru coada specificată ;

SET SYSTEM Setare anumite caracteristici sistem ;

SET SYSTEM/DIRECTORY Stabilire catalog sistem (**UFD**) pentru task-urile sistem privilegiate ;

SET SYSTEM UFD Setarea catalogului sistem (**UFD**) și a tuturor task-urilor sistem la **UIC**-ul specificat ;

SET TERMINAL Setare terminal cu atributele specificate ;

SHOW [DAY]TIME Afișare timp și dată curente ;

SHOW GROUPFLAGS Afișare indicatori de evenimente globali de grup ;

SHOW LIBRARY Afișare **UIC** biblioteci sistem ;

SHOW MEMORY Apelare **RMD** pentru afișarea dinamică a activității sistemului ;
SHOW PARTITIONS Afișare adresă și informații privind partițiile din sistem ;
SHOW SYSTEM/DIRECTORY Afișare catalog sistem curent ;
SHOW SYSTEM/LIBRARY Afișare catalog biblioteci sistem curent ;
SHOW SYSTEM Afișare informații despre sistemul curent ;
SHOW TASKS Afișare informații despre task-urile active sau instalate ;
SHOW TASKS/DYNAMIC Apelare **RMD** pentru afișarea dinamică a informațiilor relative la un task ;
SHOW USERS Afișare informații terminale introduse în sistem.

Comenzi pentru manipularea dispozitivelor și volumelor

ALLOCATE Declarare dispozitiv pentru utilizare particulară ;
ASSIGN Asociere nume logic — dispozitiv fizic, pseudo-dispozitiv sau alt dispozitiv logic ;
ASSIGN/REDIRECT Redirectare ieșire, de la un dispozitiv fizic către un alt dispozitiv fizic ;
ASSIGN/TASK Reasignare **LUN** task instalat, de la un dispozitiv fizic la alt dispozitiv fizic ;
BACKUP Apelare utilitar **BRU** pentru punere la zi și restaurare volum **FILES-11** ;
DEALLOCATE Eliberare dispozitiv alocat anterior ;
DEASSIGN Stergere asocieri nume logic dispozitiv fizic, pseudodispozitiv sau alt dispozitiv logic ;
DISMOUNT Invalidare recunoaștere volum logic ;
INITIALIZE Inițializare volum cu structură **FILES-11** ;
/UPDATE — creare antet de volum *(home block)* pentru folosire ulterioară de task-ul **F11ACP** la montarea volumului ;
MOUNT Validare recunoaștere volum logic ;
SET DEVICE Stabilire caracteristici dispozitiv ;
SHOW ASSIGNMENTS Afișare la terminal a tuturor asignărilor (locale sau globale) ;
SHOW DEVICE Afișare informații dispozitive incluse în sistem.

Comenzi de control al execuției task-urilor

ABORT Terminare forțată execuție task sau acțiune specifică unei comenzi ;
ASSIGN/REDIRECT Redirectare ieșire, de la un dispozitiv fizic la un alt dispozitiv fizic ;
ASSIGN/TASK Reasignare **LUN** al unui task instalat, de la un dispozitiv fizic la un altul ;
CANCEL Eliminare intrări din coada de ceas ;
CONTINUE Reluare execuție task anterior suspendat ;
DEBUG Apelare subsistem de depanare (numai la task-uri link-editate cu această opțiune) ;
FIX Încărcare și fixare în memorie a unui task instalat **(MIX/MIX-PLUS)** sau a unei regiuni instalate **(MIX-PLUS)** ;

HELP Afişare informaţii ajutătoare ;
INSTALL Instalare task în memorie ;
MCL Introducere comandă **MCL** ;
REMOVE Eliminare task din sistem ;
RUN Lansare în execuţie task ;
SET [NO]PARTITION Creare, eliminare partiţii în/din sistem ;
SET PRIORITY Modificare prioritate task activ ;
SHOW CLOCK-QUEUE Afişare informaţii despre task-urile curente din coada de ceas sistem ;
SHOW COMMON Afişare informaţii privind comunurile rezidente instalate ;
SHOW PARTITION Afişare informaţii partiţii existente ;
SHOW TASKS Afişare informaţii relative la task-urile instalate şi active ;
START Reluare execuţie task stopat de directiva **STOP$** ;
START/UNBLOCK Continuare execuţie task blocat anterior prin comanda **STOP/BLOCK** ;
STOP/BLOCK Blocare task aflat în execuţie ;
UNFIX Defixare task sau regiune din memorie.

Comenzi relative la terminale

BROADCAST Tipărire mesaj pe unul sau mai multe terminale ;
REQUEST Trimitere mesaj la consola operator **(CO:)** ;
SHOW ACCOUNTING/INFORMATION Tipărire informaţii curente despre sesiunea de lucru **(MIX-PLUS)** ;
SHOW TERMINAL Afişare informaţii despre terminalul specificat.

Comenzi de lucru în configuraţii multiutilizator

ALLOCATE Declarare utilizare particulară pentru dispozitivul specificat ;
LOGIN Permite accesul de la terminal la un sistem multiutilizator, stabilind anumite caracteristici ale sesiunii de lucru ;
LOGOUT Terminare sesiune de lucru într-un sistem multiutilizator.

8.10. Interpretoare de limbaje de comandă utilizator

Familia de sisteme de operare **MIX/MIX-PLUS** poate utiliza, în afara limbajelor standard de comandă **MCL** *(Mix Command Language)*, şi a limbajului extins de comandă **DCL** *(Digital Command Language)*, limbaje specifice, implementate de task-uri interpretoare de comenzi **CLI** *(Command Language Interpreters)*.

Ca interpretor al unui limbaj de comandă, un task **CLI** constituie o interfaţă specifică între utilizatorii interactivi sau aplicaţiile utilizator şi sistemul de operare. Un task **CLI** nu este neapărat un task privilegiat (ca **MCL, DCL**). El poate fi privilegiat, dacă se solicită acest lucru de o aplicaţie.

Interfaţarea cu sistemul de operare este necesară pentru sincronizarea operaţiilor de manipulare a comenzilor şi în emiterea secvenţelor de identificare. Aceasta se realizează printr-o serie de directive Monitor specifice **CLI**.

Cea mai importantă directivă este **GCCI$** (obținere comenzi pentru Interpretorul de comenzi), ce trebuie folosită de orice **CLI**. Execuția directivei **GCCI$** furnizează o linie de comandă pentru task-ul **CLI**, din coada (FIFO) menținută de sistem. Ea controlează de asemenea starea biților necesari pentru sincronizarea corectă a emiterii serviciilor de identificare și comenzile generale de manipulare.

Alte directive **CLI** sînt :

— **SCAA$** (specificare întrerupere asincronă **AST**), care validează/invalidează o tratare **AST** la nivelul **CLI**, atunci cînd apare o comandă ce urmează a fi prelucrată de task-ul CLI ;

— **GCII$** (obținere informații pentru Interpretorul de comenzi) — ce returnează unui task informații despre un terminal sau despre task-ul **CLI**.

Prelucrarea unei comenzi **CLI** necesită cunoașterea acesteia de către sistem. Aceasta se face instalînd task-ul **CLI** cu comutatorul **/CLI = YES** și inițializînd-ul cu comanda **CLI/INIT**. Acest proces creează o structură internă de date care permite sistemului să recunoască existența unui task **CLI**. În continuare se folosește comanda **SET/CLI** pentru a seta **CLI**-ul specificat drept limbaj de comandă al terminalului.

Operațiile de tip **CLI** încep atunci cînd se introduce o comandă în sistem. Comanda este prelucrată pentru început de dispecerul de comenzi (task-ul **MCR...**), care o pune într-o coadă de comenzi FIFO ce vor fi transmise către task-urile **CLI**.

CLI-ul asociat terminalului de la care s-a introdus comanda primește controlul de la sistem și citește comanda prin emiterea unei directive **GCCI$**. Directiva preia comanda din coada **CLI**-ului asociat și o copiază într-un bufer din spațiul de adresare al task-ului **CLI**. **GCCI$** poate de asemenea să returneze informații suplimentare despre terminalul de intrare asociat **CLI**.

Odată primită comanda, task-ul **CLI** o poate prelucra după cum dorește. În general o poate executa sau trece altui task pentru execuție.

Dacă task-ul **CLI** emite comanda către un alt task (care nu este **CLI**), trebuie să folosească directivele **RPOI$** (lansare în execuție și transmitere informații către un task fiu) și **SDRP$** (emisie comandă, lansare în execuție și transmitere informații).

După prelucrarea unei comenzi, **CLI**-ul trebuie să reexecute directiva **GCCI$** pentru a primi o altă comandă din coada de așteptare asociată.

Un task **CLI** poate fi instalat ca o funcție extinsă cu numele de forma ***...xxx*** (task prototip) sau cu orice nume permis, ca de exemplu ******xxx*** (task convențional).

8.11. Fișiere indirecte de comenzi

Fișierele indirecte de comenzi reprezintă un șir de comenzi, respectiv subcomenzi, adresate unui task și interpretabile de către acesta. Task-ul interpretor este de obicei o componentă a sistemului de operare **MIX**, cum ar fi : Interpretoarele de limbaje de comandă (**MCL**, **DCL**, **CLI**), Macroasamblorul, Editorul de legături, etc.

Fişierele indirecte de comenzi pot fi de două tipuri :

— *fişiere de comenzi specifice unui task* — incluzînd linii de comandă, prelucrate exclusiv de un task sistem (program utilitar, compilator, etc.) sau utilizator ;

— *fişiere de comenzi* **CLI** (fişiere referite ca indirecte), conţinînd comenzi adresare task-urilor interpretoare de comenzi **(MCL, DCL, CLI)** şi comenzi speciale (directive) adresate şi interpretate de un task sistem special, numit *Procesor de comenzi indirecte.*

Existenţa fişierelor de comenzi indirecte automatizează activitatea de exploatare a sistemului de calcul, permiţînd efectuarea unor operaţii (comenzi) în perioade de timp cu mult mai scurte.

În cadrul sistemului de operare **MIX** anumite aplicaţii sau proceduri sistem se bazează pe utilizarea exclusivă a fişierelor de comenzi indirecte. Un exemplu bun în acest sens îl constituie generarea şi reconfigurarea sistemului **MIX** care devine, în acest fel, un proces interactiv extrem de flexibil şi interesant, uşor de modificat şi actualizat.

8.11.1. Fişiere indirecte de comenzi pentru task-uri

Majoritatea task-urilor citesc şi reacţionează la comenzile conţinute într-un fişier indirect ca şi cum aceste comenzi ar fi introduse de la un terminal.

Pentru a iniţia prelucrarea unui fişier indirect, în linia de comandă se precede specificatorul de fişier de semnul „*at*" (@). Task-ul iniţiator citeşte informaţiile conţinute în fişierul indirect şi le prelucrează conform comenzilor conţinute în acesta.

De exemplu, pentru execuţia unui fişier de comenzi specifice Macroasamblorului se introduce linia de comandă :

```
) MAC  PROC.CMD
```

Un fişier de comenzi adresat unui task trebuie să conţină numai linii de comandă specifice şi valide pentru task-ul respectiv. Fişierul de comenzi nu poate conţine directive ale Procesorului de comenzi indirecte.

8.11.2. Fişiere indirecte de comenzi CLI

Fişierele indirecte de comenzi **CLI** conţin linii de comandă specifice unui task interpretor de comenzi **(MCL, DCL, CLI)** şi directive speciale ce permit controlul prelucrării fişierului de comenzi.

Fişierele indirecte de comenzi **CLI** sînt citite şi interpretate de un task sistem special, numit *Procesorul fişierelor indirecte de comenzi* **(AT.)**, care asigură :

— interpretarea comenzilor proprii (directive) ;

— transmiterea liniilor de comandă **CLI** către task-ul interpretor de comenzi ataşat terminalului curent.

Într-un fişier indirect se pot referi alte fişiere indirecte, numărul maxim de apeluri fiind limitat la patru.

Pentru a iniția execuția unui fișier indirect de comenzi **CLI**, se precede specificatorul de fișier de semnul „@", la acceptarea unei intrări solicitate pentru un task interpretor de comenzi :

```
CLI > @ CLICOM.CMD
```

8.11.3. Procesorul fișierelor indirecte de comenzi

Prelucrarea și interpretarea fișierelor indirecte de comenzi **CLI** se face de către Procesorul fișierelor indirecte de comenzi.

Directivele proprii acestui procesor sînt identificate prin prezența unui punct (.) ca prim caracter în linia de comandă. Aceste directive sînt interpretate de către Procesorul fișierelor indirecte de comenzi. Toate celelalte linii(care nu încep cu punct) sînt considerate a fi drept comenzi operator și sînt trecute task-ului interpretor de comenzi atașat terminalului curent (**MCL, DCL. CLI**). Liniile de comandă pot fi separate prin linii comentarii.

În timpul prelucrării unui fișier indirect de comenzi **CLI**, Procesorul fișierelor indirecte afișează, la terminalul curent, fiecare comandă **CLI** și liniile comentarii conținute în fișier. La activarea fișierului de comenzi cu comutatorul /**-MC** sau /**-CLI** (ex. @ *numefișier*/**-MC**), comenzile **CLI** din fișierul de comenzi se transformă în comentarii cu un semn de exclamare (!) la începutul liniei de comandă ; în acest caz comanda **CLI** este numai afișată, nu și executată.

Directivele care se adresează procesorului de fișiere indirecte de comenzi formează de fapt un *limbaj procedural* care permite efectuarea următoarelor acțiuni :

— definirea de etichete în fișierul indirect de comenzi ;

— definirea de simboluri avînd unul din tipurile : logic, numeric sau șir și atribuirea de valori sau modificarea valorilor (fără modificarea tipului) ;

— substituirea valorii unui simbol ;

— crearea și accesul la fișiere de date, prin directive aflate în fișierul indirect de comenzi ;

— controlul execuției fișierului indirect de comenzi ;

— testarea simbolurilor și condițiilor ;

— validarea sau inhibarea unui mod de operare în cadrul fișierului indirect de comenzi ;

— prelucrarea aritmetică a operațiilor ;

— manipularea șirurilor de caractere ;

— afișarea de text pe terminalul utilizator ;

— punerea de întrebări și așteptarea de răspunsuri ;

— apelarea de subrutine ;

— detectarea condițiilor de eroare ;

— execuția task-urilor în paralel sau planificat.

În sistemul de operare **MIX**, Procesorul fișierelor indirecte de comenzi este implementat sub forma unui task sistem privilegiat, mapat la Monitor ; în sistemul de operare **MIX-PLUS**, acesta este un task neprivilegiat, multi-utilizator.

Procesorul fișierelor indirecte poate fi executat pe un sistem **MIX** care suportă sau nu relații de paternitate între task-uri. Pentru motive de adaptare, el este conceput astfel încît să determine tipul sistemului sub care se execută : cu sau fără relații de partenitate între task-uri.

În cazul în care această facilitate este prezentă, (cazul sistemelor de operare **MIX/MIX-PLUS**), comanda este trecută task-ului **CLI** asociat prin directiva **SPWN$**. Task-ul apelat (interpretorul de comenzi) devine subtask sau task „fiu" pentru task-ul „tată" (procesorul de comenzi indirecte), care-i transmite o linie de comandă. **TI** :-ul subtask-ului este identic cu cel al task-ului tată.

8.11.4. Entități de lucru ale procesorului de comenzi indirecte

Entitățile de lucru ale procesorului de comenzi indirecte sînt : *simboluri ; fișiere secundare ; task-uri.*

Simbolurile — sînt definite de utilizator și pot fi testate, comparate, modificate sau substituite în cursul trecerii prin fișierul de comenzi indirecte.

Un simbol este un șir format din 1÷6 caractere ASCII, care începe în mod obligatoriu cu o literă (A÷Z) sau semnul „dolar" ($) și conține, în rest, orice caracter alfanumeric sau $.

Simbolurile admise pot fi :

— logice (cu valori adevărate sau false) ;

— numerice (octale, cu valori cuprinse între 0 și 177777, sau zecimale, cu valori cuprinse între 0 și 65.536) ;

— șiruri de caractere (cu lungimea cuprinsă între 0 și 132. caractere imprimabile, incluse între două caractere „ghilimele" (")) ;

— speciale.

Tipul unui simbol este definit prin prima directivă de atribuire a unei valori simbolului respectiv. Directivele ulterioare pot modifica valoarea simbolului, dar nu și tipul.

Unui simbol de tip *logic* i se atribuie una din valorile „adevărat" sau „fals" prin una din directivele : **.ASK, .SETT, .SETF, .SETL.**

Unui simbol de tip *numeric* i se atribuie o valoare prin una din directivele : **.ASKN, .SETN, .SETD, .SETO.**

Unui simbol de tip *șir* i se atribuie o valoare prin una din directivele : **.ASKS, .SETS, .READ.**

Atunci cînd se definește un simbol, procesorul fișierelor de comenzi indirecte creează o intrare într-o tabelă internă de simboluri. Cu ajutorul directivelor **.BEGIN** și **.END** se pot crea blocuri în cadrul unui fișier de comenzi. Un astfel de bloc izolează definițiile de simboluri locale și etichetele ; la întîlnirea directivei **.END** aceste simboluri sînt șterse din tabela de simboluri, ceea ce permite economisirea spațiului necesar pentru această tabelă.

În absența blocurilor **.BEGIN, .END,** simbolurile definite într-un fișier de comenzi se păstrează pe tot timpul execuției.

Pentru ca un simbol să devină global, este necesar ca el să înceapă cu caracterul **$** și fișierul de comenzi indirecte să fie în modul de operare global (prin directiva **.ENABLE**). În acest caz, un simbol global va fi păstrat în tabela internă de simboluri pe tot timpul execuției, chiar dacă face parte dintr-un bloc **.BEGIN, .END.**

Fișiere secundare — constituie fișiere de ieșire pentru utilizator, conținînd directive ale procesorului de comenzi indirecte și comenzi ale procesorului de comenzi operator.

Task-urile sistem sau utilizator — constituie elemente de comparare și decizie ale directivelor procesorului de comenzi indirecte.

Tipuri de expresii folosite în procesorul comenzilor indirecte

Un simbol numeric poate fi combinat cu alte simboluri numerice și cu constante, prin intermediul unor operatori aritmetici sau logici, pentru a forma expresii numerice.

Operatorii aritmetici care se pot utiliza sînt :

— adunare : +
— scădere : —
— înmulțire : ×
— împărțire : /

Operatorii logici care se pot utiliza sînt :

— sau exclusiv : !
— și logic : &

În interiorul unei expresii nu sînt permise caractere *spațiu (blank)* sau *tab*. Evaluarea unei expresii numerice se face de la stînga la dreapta, toți operatorii avînd aceeași prioritate. Schimbarea ordinei de evaluare se poate face prin utilizarea de paranteze.

Dacă toți operanzii unei expresii sînt octali, atunci tipul expresiei este octal ; în caz contrar, tipul expresiei este zecimal.

Operatorii logici acționează bit cu bit asupra valorii binare a operanzilor lor.

Expresiile numerice pot apare în directivele **.IF** și **.SETN** pe locul celui de-al doilea operand, în directivele **.ASKN** pentru a indica domeniul argumentelor și în directivele **.EXIT** și **.STOP** pentru reprezentarea stării de ieșire.

8.11.5. Expresii de tip șir

Un simbol de tip *șir* poate lua ca valoare o constantă de tip șir de caractere.

Atribuirea de valori pentru simbolurile de tip șir se face prin directivele **.SETS** sau **.ASKS**.

În directivele **.IF** și **.SETS**, se pot folosi pe locul celui de-al doilea operand, subșirurile de caractere, a căror formă generală este : *șir* [*început* : *sfîrșit*].

Cu ajutorul simbolurilor de tip șir, a subșirurilor și a constantelor de tip șir se pot forma expresii de tip șir. Operatorul care se folosește în aceste expresii este operatorul de *concatenare*, notat prin caracterul „*plus*“ (+).

Expresiile de tip șir se pot utiliza în directivele **.SETS** și **.IF** pe locul celui de-al doilea operand, dacă primul operand este un simbol de tip șir.

Delimitatorii pentru expresii și simboluri de tip șir sînt reprezentați de caracterele diez (#) și ghilimele (“).

8.11.6. Substituirea valorilor pentru simboluri

Cu ajutorul directivei **.ENABLE**, se poate valida pentru un fişier de comenzi indirecte operarea în mod *substituţie* (**SUBSTITUTION**).

În acest caz, procesorul de comenzi indirecte va înlocui orice apariţie a unui nume de simbol inclus între caracterele apostrof (') (de forma *'simbol'*) cu valoarea simbolului respectiv, obţinută ca urmare a execuţiei ultimei directive de atribuire pentru acest simbol.

Substituirea valorii unui simbol se poate face pentru oricare tip : logic, numeric sau şir de caractere.

Atunci cînd modul substituţie este validat, includerea unui apostrof într-o linie de comandă se face prin dublarea sa, avînd în vedere că un singur caracter apostrof este utilizat pentru a marca începutul sau sfîrşitul unei construcţii de forma *'simbol'*.

8.11.7. Simboluri speciale

În procesorul fişierelor de comenzi indirecte sînt definite cîteva simboluri speciale, dependente de caracteristicile sistemului gazdă şi de răspunsurile date la întrebările puse în decursul execuţiei fişierului de comenzi. Există simboluri speciale pentru toate tipurile admise : logic, numeric şi şir de caractere.

Aceste simboluri pot fi testate, comparate şi substituite. Formatul lor este acelaşi pentru toate tipurile :

⟨*nume simbol*⟩

Un caz particular îl reprezintă simbolurile utilizate în transmiterea parametrilor unei linii de comandă către Procesorul de comenzi indirecte. Ele contravin regulii ⟨*simbol*⟩, avînd denumiri speciale.

Simboluri speciale de tip logic

Există următoarele simboluri speciale de tip logic care iau una din valorile „adevărat" sau „fals" :

⟨**ALTMOD**⟩ Răspunsul s-a terminat cu caracterul **ALTMODE** sau **ESCAPE** ;

⟨**ALPHAN**⟩ Şirul conţine numai caractere alfanumerice sau este şirul vid ;

⟨**BASLIN**⟩ Sistemul de operare curent este *sistem baseline* ;

⟨**DEFAUL**⟩ Răspunsul a fost caracterul „retur de car" sau a apărut „time-out" ;

⟨**ESCAPE**⟩ Analog cu ⟨**ALTMOD**⟩ ;

⟨**ERSEEN**⟩ Adevărat, dacă una din următoarele este adevărată :

FILERR are valoare negativă ;

— **EXSTAT** are valoare mai mică decît **WARNING** ;

— **EOF** are valoare adevărată ;

— **ERRNUM** este diferit de 0 ;

— dacă se foloseşte linia de comandă **.SETT ERSEEN.**

⟨**EOF**⟩ În citirea (**.READ**) unui fișier de date deschis în citirea (**.OPENR**) sau cînd răspunsul la o directivă **.ASK** este **CTRL/Z**;

⟨**FALSE**⟩ Constantă logică pentru compararea directivelor **.IF** sau ca răspuns negativ la directivele **.ASK**;

⟨**LOCAL**⟩ Terminalul de la care se execută Procesor de comenzi indirecte (**TI:**) este local;

⟨**MAPPED**⟩ Procesorul de comenzi indirecte se execută pe un sistem mapat;

⟨**NUMBER**⟩ Ultimul șir are numai caractere numerice sau este nul;

⟨**OCTAL**⟩ Răspunsul la ultima directivă sau baza de numerație a simbolului este un număr octal;

⟨**PRIVIL**⟩ Utilizatorul este privilegiat;

⟨**RAD50**⟩ Conține numai caractere Radix-50 sau este șirul vid;

⟨**RSDIID**⟩ Compatibilitate cu alte sisteme; valoarea sa este „fals";

⟨**TIMEOUT**⟩ Este activat modul timeout și intervalul de timp specificat în ultima directivă **.ASK** este depășit.

Simboluri speciale de tip numeric

Există următoarele simboluri de tip numeric:

⟨**ERRCTL**⟩ Este o mască de opt biți pentru controlul modului de prelucrare al erorilor. Valoarea 200(8) semnifică netipărirea mesajelor de eroare.

⟨**ERRNUM**⟩ Numărul clasei unei erori prelucrate de **CLI**;

⟨**ERRSEV**⟩ Masca de erori grave asociată unei erori prelucrate de **CLI**;

⟨**EXSTAT**⟩ Exprimă starea cu care s-a ieșit din ultima directivă a procesorului de comenzi indirecte (**0, 1, 2, 4**);

⟨**FILERR**⟩ Are ca valoare (în octal) codul de stare returnat în urma verificării existenței unui fișier;

⟨**FILER2**⟩ Codul secundar de eroare **FCS** returnat de ultima directivă **FCS** executată (numai pentru **MIX-PLUS**);

⟨**FORATT**⟩ Număr octal ce reprezintă atributele fișierelor folosite pentru bazele de date;

⟨**MEMSIZ**⟩ Are ca valoare dimensiunea memoriei interne;

⟨**NOSTAT**⟩ Simbol literal numeric care verifică valorile returnate de task;

⟨**SPACE**⟩ Are ca valoare (în octal) numărul de octeți liberi în tabela internă de simboluri a procesorului de comenzi indirecte;

⟨**STRLEN**⟩ Are ca valoare lungimea șirului introdus la ultima directivă a procesorului de comenzi indirecte;

⟨**SYSUNIT**⟩ Are ca valoare numărul dispozitivului sistem (**SY:**);

⟨**SYMTYP**⟩ Codul tipului de simbol testat de directiva **CLI** (*0, 2, 4*);

⟨**SYSTEM**⟩ Compatibilitatea cu alte sisteme; valoarea sa este *1* pentru **MIX** și *6* pentru **MIX-PLUS**;

⟨**TICLPP**⟩ Lungimea curentă (în număr de linii) a paginii de intrare terminal (implicit 24(10));

⟨**TICWID**⟩ Mărimea liniei de intrare terminal (implicit 80(10));

⟨**TISPED**⟩ Viteza de transmisie la terminalul de intrare;

⟨**TITYPE**⟩ Tipul terminalului de la care se execută task-ul **CLI** asociat.

Simboluri speciale de tip şir

Există următoarele simboluri speciale de tip şir :

⟨**ACCOUN**⟩	Conţine informaţii preluate din Blocul de contabilizare „informaţii utilizator (**UAB**) (numai pentru **MIX-PLUS**) ;
⟨**CONFIG**⟩	Conţine parametrii impliciţi specificaţi la construcţia task-ului **CLI** asociat ;
⟨**CLI**⟩	Acronimul (şir de 3 caractere) task-ului interpretor de comenzi curent **CLI (MCL, MCR, DCL)** ;
⟨**DATE**⟩	Data curentă sub forma : *zz-lll-aa* ;
⟨**DIRECT**⟩	Şir ce descrie numele catalogului implicit ;
⟨**EXSTRI**⟩	Şir care returnează rezultatele dintr-un fişier de comenzi apelant ;
⟨**FILATR**⟩	Conţine cele 7 cuvinte atribute conţinute în Blocul de descriere al unui fişier (**FDB**), corespunzător ultimului fişier specificat în directiva **.OPEN** ;
⟨**FMASK**⟩	Conţine valori octale reprezentînd cele 5 cuvinte de caracteristici sistem, separate prin virgulă ;
⟨**FILSPC**⟩	Dispozitivul fizic asociat unui dispozitiv logic, sub forma **DD***nnn* ;
⟨**LIBUIC**⟩	**UIC**-ul bibliotecii task-ului neprivilegiat curent, sub forma : [*ggg, mmm*] ;
⟨**LOGDEV**⟩	Numele dispozitivului de intrare şi numărul corespunzător utilizatorului introdus (logat) în sistem (în sisteme cu protecţie multiutilizator) ;
⟨**LOGUIC**⟩	**UIC**-ul utilizatorului curent sub forma [*ggg, mmm*] ;
⟨**NETUIC**⟩	Şir ce conţine valoarea **UIC**-ului sistem folosit pentru identificarea task-urilor reţelei **MININET/MIX** ;
⟨**NETNOD**⟩	Numele nodului sistemului curent ;
⟨**NXTSYM**⟩	Afişează conţinutul curent al tabelelor de simboluri locale şi globale ;
⟨**SYDISC**⟩	Mnemonica (2 caractere) a dispozitivului sistem **SY:** sub forma **DD** ;
⟨**SYSDEV**⟩	Numele dispozitivului de intrare/ieşire fizic de pe care este încărcat sistemul de operare ;
⟨**SYSID**⟩	Numărul de identificare al sistemului de operare (1 ÷ 6 caractere) ;
⟨**SYSUIC**⟩	**UIC**-ul sistem, sub forma : [*ggg, mmm*] ;
⟨**SYTYP**⟩	Şir ce descrie tipul de sistem (de exemplu **MIX-PLUS**) ;
⟨**TIME**⟩	Ora curentă sub forma *hh* : *mm* : *ss* ;
⟨**UIC**⟩	**UIC**-ul curent, sub forma [*ggg, mmm*] ;
⟨**VERSN**⟩	Numărul curent de versiune a sistemului de operare (3 caractere).

Simboluri speciale pentru transmiterea parametrilor unei linii de comandă

Procesorul de comenzi indirecte acceptă o linie de comandă, cu formatul general :

@*numefişiercomenzi Parametru 1 Parametru 2 . . . Parametru 9.*

La primirea unei astfel de comenzi, următoarele simboluri primesc valori de tip şir :

P0	Valoarea sa este şirul de caractere *'@ numefişiercomenzi'* ;
P1	Valoarea sa este şirul de caractere *'Parametru 1'* ;
P2, P3, P4, P5	Valorile lor sînt şirurile de caractere asociate parametrilor
P6, P7, P8, P9	corespunzători ;
COMMAN	Valoarea sa este şirul de caractere cuprinzînd întreaga linie de comandă, aşa cum a fost transmisă de **MCL** către Procesorul de comenzi indirecte.

8.11.8. **Directive pentru Procesorul comenzilor indirecte**

Procesorul comenzilor indirecte suportă un *set propriu de directive.* Aceste directive sînt următoarele :

a) **definire de etichete**

.etich :	Atribuirea unui nume liniei de comandă pe care apare, pentru a permite referirea ulterioară a acestei linii ;

b) **definire de simboluri, atribuire sau modificare de valori**

.ASK	Defineşte sau redefineşte un simbol logic şi atribuie simbolului o valoare logică (adevărat sau fals) ;
.ASKN	Defineşte sau redefineşte un simbol numeric şi atribuie simbolului o valoare numerică ;
.ASKS	Defineşte sau redefineşte un simbol de tip şir şi atribuie simbolului o valoare constînd dintr-un şir ASCII ;
.ERASE	Şterge definiţiile de simboluri locale şi globale ;
.SETT/.SETF	Defineşte sau redefineşte un simbol logic şi îi atribuie o valoare logică (adevărat/fals) ;
.SETN	Defineşte sau redefineşte un simbol numeric şi îi atribuie o valoare numerică ;
.SETD	Defineşte sau redefineşte tipul unui simbol numeric, ca zecimal ;
.SET0	Defineşte sau redefineşte tipul unui simbol numeric ca octal ;
.SETS	Defineşte sau redefineşte un simbol de tip şir şi îi atribuie ca valoare un şir de caractere ;
.SETL	Defineşte sau redefineşte simbolul logic şi dă simbolului valoarea „adevărat“ sau „fals“.

c) **creare, acces şi verificare fişiere**

.OPEN	Creează şi deschide un fişier de date ; dacă fişierul există deja, se creează o nouă versiune a acestuia şi apoi se deschide ;
.OPENA	Deschide un fişier de date existent şi adaugă noi date ; această operaţie nu duce la modificarea versiunii fişierului ;
.OPENR	Deschide un fişier de date existent pentru a fi citit linie cu linie ;

.READ	Citește o singură linie de date dintr-un fișier specificat cu acces numai în citire ;
.DATA	Specifică o singură linie de date care urmează să se introducă într-un fișier de date ;
.CLOSE	Închide fișierul de date, deschis anterior ;
.TESTFILE	Verificarea existenței unui fișier cu identificarea dispozitivului fizic asociat ;
.CHAIN	Închide fișierul de comenzi indirecte curent și se trece la prelucrarea comenzilor dintr-un alt fișier de comenzi indirecte.

d) controlul execuției fișierului de comenzi

.BEGIN	Stabilește începutul unui bloc **.BEGIN/.END** ;
.END	Semnalează sfîrșitul unui bloc **.BEGIN/.END** ;
.GOTI	Provoacă un salt la o etichetă din cadrul fișierului de comenzi ;
.GOSUB	Provoacă apelul unei subrutine aflată la o etichetă în cadrul fișierului de comenzi ;
.RETURN	Provoacă ieșirea dintr-o subrutină și reluarea execuției fișierului de comenzi de la linia care urmează după linia de apel ;
.EXIT	Termină prelucrarea blocului **.BEGIN/.END** (dacă se găsește în interiorul unui astfel de bloc) sau a fișierului de comenzi curent, redă controlul nivelului precedent și (opțional) setează valoarea simbolului special **⟨EXSTAT⟩** ;
.ONERR	Provoacă saltul la o etichetă în cazul detectării unei condiții de eroare specifice pentru Procerosul de comenzi indirecte ;
.STOP	Termină prelucrarea fișierului de comenzi indirecte și (opțional) setează starea de ieșire a Procesorului de comenzi indirecte ;

e) teste logice

.IF	Determină dacă un simbol satisface sau nu, una din mai multe condiții posibile ;
·IFACT/.IFNACT	Determină dacă un task este sau nu activ ;
.IFDF/.IFNDF	Determină dacă un simbol este sau nu definit ;
.IFINS/.IFINS	Determină dacă un task este sau nu instalat în sistem ;
.IFLOA/.IFNLOA	Determină dacă un driver este sau nu încărcat ;
.IFT/.IFF	Determină dacă un simbol de tip logic are valoarea adevărat sau fals ;
.TEST	Testează lungimea unui simbol de tip șir ;
.IFDISABLED	Testează opțiunea **.ENABLE/.DISABLE** ;
.TESTDEVICE	Returnează date privind dispozitivul de intrare /ieșire specificat ;
.TESTPARTITION	Returnează date privind partiția specificată ;
.TESTFILE	Testează dacă un fișier există sau nu.

f) validarea sau inhibarea unui mod de operare

.ENABLE Validează legalitatea unuia dintre modurile de operare posibile pentru fișierul de comenzi indirecte : substituție, date, simboluri globale, caractere minuscule, recunoașterea unui caracter „escape", controlul ecoului ;

.DISABLE Inhibă unul dintre modurile de operare pentru fișierul de comenzi indirecte, enumerate la **.ENABLE.**

g) incrementarea/decrementarea simbolurilor numerice

.INC Incrementează cu 1 valoarea unui simbol numeric ;
.DEC Decrementează cu 1 valoarea unui simbol numeric.

h) controlul timpului de execuție și lansarea de task-uri în paralel

DELAY Suspendă execuția unui fișier de comenzi indirecte pentru o perioadă specificată de timp ;

.PAUSE Oprește execuția procesorului fișierelor de comenzi indirecte permițînd o acțiune utilizator ;

.WAIT Așteaptă pînă la terminarea execuției unui task specificat ;

.XQT Inițiază execuția unui task, căruia îi transmite o linie de comandă, continuîndu-se execuția fișierului de comenzi indirecte în paralel cu cea a task-ului inițiat.

i) analiza unei linii de comandă

.PARSE Parsarea unei linii de comandă.

j) Scop special (numai în MIX-PLUS)

FORM Activează ca interfață pentru driverul de formate **FMS** ;
.TRANSLATE Translatează (multiplu) nume logice.

8.11.9. Prelucrarea indirectă pe mai multe nivele

Dintr-un fișier de comenzi indirecte, utilizatorul are posibilitatea de a apela un nou fișier de comenzi, utilizînd o comandă de forma :

@ specificator de fișier

Numărul maxim de nivele de indirectare admis este de patru.

Simbolurile definite la un nivel de prelucrrare indirectă se consideră simboluri locale acestui nivel. La trecerea pe un alt nivel, aceste simboluri sînt mascate, astfel încît vor fi disponibile numai simbolurile definite la noul nivel. Cînd prelucrarea noului nivel se termină, simbolurile definite pe acest nivel sînt pierdute și simbolurile definite pe vechiul nivel devin din nou disponibile.

Totuşi utilizatorul are posibilitatea de a păstra simbolurile definite la un nivel şi pentru nivelele activate din acesta, prin utilizarea directivei **.ENABLE GLOBAL.**

Terminarea prelucrării unui nivel de indirectare se face la întîlnirea unei directive **.EXIT** sau la epuizarea comenzilor din fişier.

8.12. Subsistemul de gestiune a cozilor (QMG)

Subsistemul de gestiune a cozilor (**QMG**) este un set de programe care asigură tipărirea asincronă, cu cozi multiple de intrare/ieşire (**MIX/MIX-PLUS**) şi prelucrarea în loturi *(batch)* (**MIX-PLUS**) a job-urilor.

Pentru aceasta, **QMG** recepţionează cerinţele de tipărire şi prelucrare în loturi a job-urilor, introduce în cozi cererile şi asigură ca aceste job-uri să fie prelucrate de unul din următoarele două tipuri de procesoare :

— *procesoare batch*, de prelucrare în loturi a job-urilor, care sînt prezentate subsistemului printr-o comandă **SUBMIT** ;

— *procesoare de ieşire*, ce controlează transferul datelor pe dispozitivele de tipărire, la emiterea explicită a comenzii **PRINT** către subsistem sau implicit, cînd diverse task-uri sistem sau de aplicaţie crează mape, listing-uri sau alte fişiere cu înregistrări ce pot fi tipărite.

Toate sistemele care execută prelucrări în loturi conţin o coadă implicită numită **BATCH**, iar procesoarele de ieşire o coadă numită **PRINT**.

Subsistemul **QMG** înregistrează toate cozile sale în fişierul **SPO : [1,7] QUEUE.SYS** care se află pe un disc de masă. De aici rezultă că nici un job nu se va pierde la întreruperea dintr-un motiv oarecare a funcţionării sistemului.

Fişierul **[1,2] QMGSTART.CMD** de pe discul sistem conţine comenzile necesare pentru lansarea subsistemului **QMG** cu un procesor de tipărire şi un procesor de prelucrare în loturi. Comenzile din acest fişier pot fi executate şi ca parte a procedurii de iniţializare sistem **[1,2] STARTUP.CMD**.

Un utilizator poate modifica fişierul de lansare astfel încît acesta să corespundă cerinţelor individuale ale sistemului de calcul, sau poate să introducă comenzi **QMG** de la terminal, în mod interactiv, pentru lansarea şi exploatarea acestuia.

8.12.1. Procesoarele de ieşire

Cînd o imprimantă sau un alt dispozitiv de tipărire este sub controlul subsistemului **QMG**, se spune că reprezintă un dispozitiv capturat „*spooled*". Aceste dispozitive trebuie iniţializate cu anumite atribute specifice procesului de tipărire.

Atributele sînt specificate de calificatorii comenzii **PRINT**.

Dispozitivele de tipărire sînt controlate de subsistemul **QMG** prin intermediul task-urilor procesoare de tipărire, ce devin „proprietari" ai acestor dispozitive.

Orice dispozitiv de tipărire „spooled" reprezintă un ansamblu dintre un dispozitiv hardware şi un procesor asociat.

Realizarea acestui ansamblu, între dispozitivele hardware de tipărire şi task-urile procesoare, se face în cadrul subsistemului **QMG** la „iniţializarea" cozilor, de către inginerul de sistem, sau prin utilizarea comenzii **PRINT.**

O coadă poate fi asociată unuia sau mai multor dispozitive, după cum un singur dispozitiv poate avea asociată lui una sau mai multe cozi. Un sistem poate avea cel mult 15 dispozitive de tipărire.

Cererile de tipărire, reunite în cozi, sînt extrase din cozi într-o manieră secvenţială.

Operaţiile executate de subsistemul **QMG** sînt controlate de un număr de comenzi **DCL.** Aceste comenzi urmează să fie prezentate pe scurt în paragraful următor.

8.12.2. Sumar al comenzilor QMG

ASSIGN	Stabilire cale de legătură coadă-procesor ;
DEASSIGN	Eliminare cale de legătură dintre coadă şi procesor sau dispozitiv ;
DELETE/PROCESSOR	Ştergere procesor batch sau de ieşire ;
DELETE	Ştergere coadă, intrare în coadă sau nume job din coadă ;
HOLD	Blocare job în coadă pînă la eliminarea lui explicită ;
INITIALIZE/QUEUE	Creare, denumire şi startare coadă ;
INITIALIZE/PROCESSOR	Iniţializare procesor ;
PRINT	Introducere job(uri) în coada unui procesor de ieşire ;
RELEASE	Eliberare intrare sau job dintr-o coadă ;
SET QUEUE	Modificare caracteristici pentru job-urile din cozi ;
SHOW PROCESSOR	Afişare nume, stări, cozi atribuite procesoarelor batch sau de ieşire ;
SHOW QUEUE	Afişare informaţii despre job-urile de tipărire şi prelucrare în loturi ;
START/QUEUE	Startare coadă stopată anterior ;
START/QUEUE/MANAGER	Startare **QMG** pentru iniţializarea cozilor implicite şi creare fişier **LB0:[1,7]QUEUE.SYS** ;
START/PROCESSOR	Startare procesor batch sau de ieşire ;
STOP/ABORT	Eliminare job activ de la un procesor dat ;
STOP/QUEUE	Stopare coadă (cozi) ;
STOP/QUEUE/MANAGER	Oprire **QMG** şi eliminare procesoare ;
STOP/PROCESSOR	Stopare procesor (procesoare) ;
SUBMIT	Introducere job(uri) în coada procesorului „batch".

8.13. Prelucrarea în loturi (MIX-PLUS)

Prelucrarea în loturi reprezintă o metodă de transmitere a comenzilor către sistemul de operare fără intervenţia utilizatorului. Astfel lucrul poate fi planificat şi procesat în modul „*background*“ (modul cel mai puţin prioritar). Prelucrarea în loturi permite crearea de job-uri, execuţia fazelor job-ului şi includerea de date în joburi „*batch*“. Această prelucrare se specifică subsistemului **QMG** prin comanda **SUBMIT** care conţine un specificator pentru unul sau mai multe job-uri „batch“ utilizator sau fişiere, pentru a fi prelucrate. Task-ul de gestiune a cozilor **QMG** inserează un număr de intrare într-o coadă a procesorului „batch“ pentru fiecare comandă **SUBMIT**. Acest număr reprezintă un job „batch“ pentru **QMG**. Apoi task-ul de gestiune a cozilor **QMG** trece controlul task-ului procesor asociat. Procesorul efectuează execuţia fiecărui job „batch“ pasînd procesorului **DCL** comenzi individuale.

Cînd un procesor „batch“ este iniţializat, el creează un terminal virtual şi lansează job-urile în execuţie de la acel terminal. De asemenea job-ul „batch“ trebuie să îndeplinească rolul utilizatorului la terminal ; adică, job-ul „batch“ trebuie să se introducă în sistem, să emită comenzi, să dea informaţii ca răspuns la prompteri, să furnizeze datele necesare programelor pe care le execută, să răspundă la erori şi să se deconecteze de la sistem. Mai mult, pentru că terminalul virtual nu are asocieri logice în momentul creării lui, aceste asocieri trebuie făcute de job-ul „batch“ însuşi. Terminalul virtual are acelaşi **UIC** implicit ca şi terminalul de la care a fost introdusă comanda **SUBMIT**.

Comenzile **QMG SHOW QUEUE** furnizează informaţii despre job-urile batch din cozile procesoarelor. Alte comenzi **QMG** asigură menţinerea job-urilor în cozi, sau ştergerea job-urilor din cozi, toate de la terminalul utilizator.

Un job „batch“ poate să execute aproape orice operaţie pe care un utilizator o poate specifica de la terminal interactiv, inclusiv compilare, asamblare, editare de legături sau execuţie de task utilizator. Un job „batch“ poate fi supus prelucrării în orice moment. Job-urile „batch“ pot fi de asemenea lansate în execuţie cu întîrziere. Utilizatorul poate să transmită comanda **SUBMIT** de la un terminal, apoi să întrerupă sesiunea de lucru sau să execute altceva fără urmărirea job-ului „batch“.

8.13.1. Job-uri „batch“

Un job „batch“ utilizator este un set de comenzi şi date în scopul prelucrării de către un procesor batch. Comenzile „batch“ sînt un subset al comenzilor **DCL**, identificate de semnul $ ca prim caracter. Cînd procesorul întîlneşte o comandă „batch“ pentru prelucrare, comanda va fi transmisă procesorului **DCL**. Data constituie orice informaţie cerută ca intrare pentru un job „batch“. Aceasta este o definiţie specifică prelucrării în loturi. Adică, data include informaţia ce urmează să fie prelucrată de job, răspunde la prompteri din comenzile job-ului şi răspunde la comenzile din fişierele indirecte de comenzi. De fapt, tot ceea ce un utilizator poate să introducă de la terminal, în afară de comenzi, constituie dată.

Liniile de date (care nu sînt identificate cu $ în prima poziţie) urmează comanda care necesită date. Procesorul „batch“ interpretează orice linie care

începe cu $ drept linie de comandă și orice linie fără $ în prima poziție, drept dată.

Un job „batch“ trebuie să fie startat cu o comandă **JOB**, care leagă job-ul la terminalul virtual. El trebuie să se încheie cu o comandă **EOJ** care întrerupe legătura terminal virtual. — job. Toate celelalte comenzi trebuie să apară între acestea două.

Comenzile job-ului „batch“ pot conține etichete (1—6 caractere). Etichetele permit unei linii de comandă să fie referită în scopul trasferării controlului.

Job-urile „batch“ pot fi programate astfel încît să răspundă codurilor de retur de stare emise de diferite task-uri sistem, la terminarea execuției unui job din șirul de job-uri „batch“. Task-urile sistem își pot termina execuția cu unul din următoarele patru coduri de stare : **SUCCES** (rezultatul este cel așteptat) ; **ANTENTIONARE** (task-ul s-a executat cu posibile erori) ; **EROARE** (rezultat diferit de cel prevăzut) ; **EROARE GRAVA** (eroare fatală sau **ABORT**).

Procesorului „batch“ îi pot fi transmise comenzi speciale de control al secvenței pentru a specifica răspunsul dorit la codurile de retur de stare.

Răspunsurile posibile sînt următoarele : stopare job ; continuare job ; transfer la alt punct de intrare în job.

O secvență specială de control poate fi stabilită pentru întregul job sau pentru o singură comandă din job-ul „batch“, folosind comenzile **ON** și **IF**. În job-uri „batch“ pot fi definite și folosite etichete.

Utilizatorii pot apela fișiere de comenzi indirecte din interiorul job-urilor „batch“ și pot transfera informații de la job-uri „batch“ fișierelor de comenzi indirecte. Job-urile „batch“ pot fi supuse prelucrării din alte job-uri „batch“ sau din fișiere de comenzi indirecte.

Comenzile de job „batch“ pot fi continuate pe două sau mai multe linii, dar pentru aceasta ultimul caracter al liniei curente trebuie să fie semnul „—“, care indică continuarea liniei respective pe următoarea linie. Liniile de continuare nu se identifică cu semnul $ și nu pot fi etichetate.

Job-urile „batch“ pot fi comentate cu un text corespunzător trecut după semnul exclamării (!). Comentariile apar în job-ul „batch“ dar nu sînt prelucrate.

8.13.2. Sumar al comenzilor „batch“

$ALLOCATE	Alocare prima unitate disponibilă și asociere dispozitiv — nume logic specificat ;
$CONTINUE	Continuare prelucrare job ;
$CREATE	Creare fișier ;
$DATA	Marcare început de bloc de date ;
$DISMOUNT	Deconectare volum (demontare) de la sistem ;
$EOD	Sfîrșit bloc de date ;
$EOJ	Sfîrșit job „batch“ utilizator și deconectare terminal virtual de la job-ul „batch“ ;
$GOTO	Transferare control unei linii etichetate ;
$IF *sc* **THEN CONTINUE**	Testare cod de stare returnat în urma comenzii precedente și, eventual, continuarea prelucrării ;

$IF *sc* **THEN GOTO**	Testare cod de stare returnat și, eventual, salt la linia etichetată ;
$IF *sc* **THEN STOP**	Testare cod de stare returnat și, eventual, oprire execuție job ;
$JOB	Marcare început de job și legare job la terminalul virtual ;
$MOUNT	Conectare logică (montare) a unui volum la sistem;
$ON *sc* **THEN CONTINUE**	Căutare de către procesor a unui anumit od de stare (*sc*), în corpul job-ului, și dacă acesta a fost găsit continuă prelucrarea de la o linie etichetată ;
$ON *sc* **THEN STOP**	Căutare de către procesor a unui anumit cod de stare (*sc*), în corpul job-ului și dacă acesta a fost găsit oprire prelucrare job ;
$SET [NO]ON	Admitere (interzicere) comanda **$ON** ;
$STOP	Stopare prelucrare job „batch“ ;
$SUBMIT	Lansare job drept job „batch“ curent ;
$TYPE	Tipărire fișier la terminalul utilizator ;
$@*indirect***.CMD**	Apelare fișier de comenzi indirecte dintr-un job „batch“
$!	Comentariu..

8.14. Încărcarea și inițializarea sistemului

Dată fiind marea varietate a suporturilor de distribuție **MIX** există o serie de proceduri generale sau speciale de încărcare a sistemului în memorie.

Sistemul de operare **MIX** este distribuit pe : bandă perforată **(MIX/RT)**, bandă magnetică (800 bpi sau 1 600 bpi), **(MIX-RT)**, disc magnetic (flexibil, încasetat, amovibil, fix) **(MIX/MIX-PLUS/MIX-RT)**.

Încărcarea unui astfel de suport în memorie se face utilizînd :

— emulatorul panoului de comandă al minicalculatorului ;

— încărcătoare primare specifice fiecărui tip de suport, în cazul în care lipsește emulatorul panoului de comandă ;

— sisteme de salvare/restaurare autonome furnizate de **MIX (MIXBRU, MIXDSC)**.

După încărcarea în memorie, procedura de inițializare sistem reconfigurează o serie de parametri relativi la configurația curentă de lucru :

— numărul și starea dispozitivelor de intrare/ieșire conectate la sistem ;

— dimensiunea reală a memoriei ;

— tipul de unitate centrală (cu/fără unitate de relocare și protecție a memoriei) ;

— ceasul sistem (cuplat pe frecvența tensiunii de alimentare sau programabil) ;

— tipul și caracteristicile variantei de sistem **MIX**.

Reconfigurarea dinamică a parametrilor importanți ai sistemului de operare **MIX**, în momentul inițializării acestuia, asigură adaptarea rapidă a sistemului la cerințele aplicației curente, eliminînd procedurile greoaie și costisitoare de regenerare sau reconfigurare operator.

9

Sistemul de intrare/ieșire MIX

9.1. Filozofia sistemului de intrare/ieșire

În cadrul sistemului de operare **MIX**, sistemul de intrare/ieșire reprezintă o ierarhie deschisă (în sensul că poate fi accesată la oricare din nivelele sale), cu mai multe niveluri software și hardware —, ce au o responsabilitate precisă în prelucrarea intrărilor/ieșirilor și gestiunea fișierelor :

— *Task-urile de aplicație* (sistem sau utilizator) — inițiază, direct sau indirect, operațiile de intrare/ieșire, gestiunea fișierelor și a datelor ;

— *Mecanismul intrărilor/ieșirilor din sistemul de operare* — controlează nivelul hardware, asigură interfața componentelor sistemului de operare cu acesta, include servicii de intrare /ieșire și gestiune a fișierelor și permite interfațarea cu task-urile utilizator ;

— *Nivelul hardware* — constituit din controloarele de dispozitiv și unitățile atașate acestora (pe care se păstrează informațiile de intrare/ieșire).

Toate cererile de intrare/ieșire sînt inițiate prin intermediul directivelor Monitor **QIO$** și **QIOW$**. Aceste directive pot fi lansate direct de către task sau indirect prin intermediul :

— subsistemelor de execuție (*run-time systems*), asociate limbajelor de nivel înalt ;

— sistemului de gestiune a fișierelor, ce asigură :

— servicii de acces la fișiere (**FCS-File Control Services**) ;

— servicii de gestiune a înregistrărilor (**RMS-Record Management Services**).

Indiferent de modalitatea specifică de lansare, directiva Monitor **QIO** constituie singura metodă prin care se pot solicita servicii de intrare/ieșire din partea dispozitivelor de intrare/ieșire.

Relațiile existente între task-urile de aplicație (sistem și utilizator) și mecanismul de tratare a intrărilor/ieșirilor, în cadrul familiei de sisteme de operare **MIX**, este reprezentat, schematic, în figura 9.1.

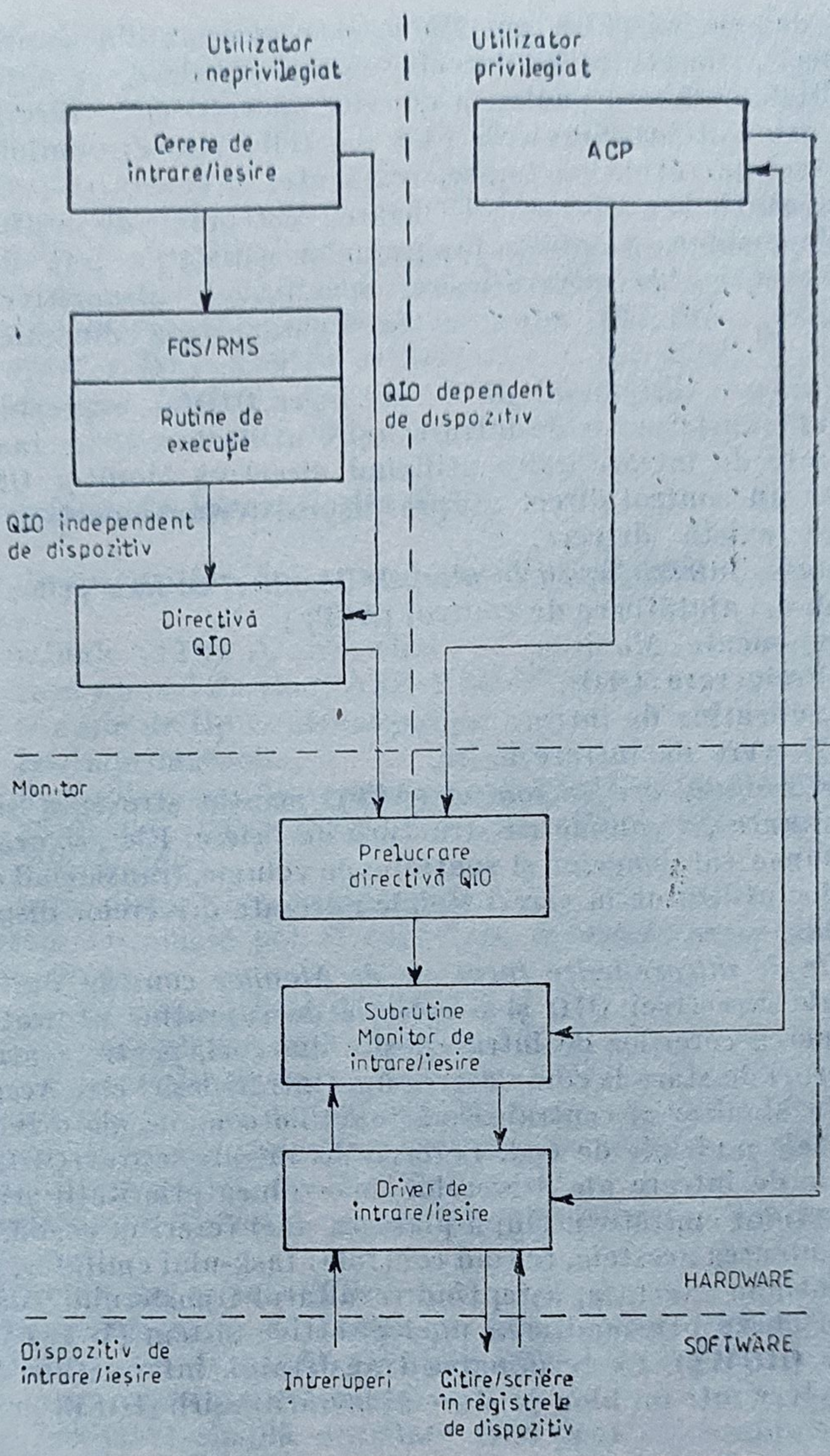


Fig. 9.1. Diagrama sistemului de intrare/ieșire **MIX**

Nivelele mecanismului de intrare/ieșire din sistemul de operare au următoarele caracteristici specifice :

a) *Subsistemele de execuție* — reprezintă seturi de rutine (în cod obiect), plasate în bibliotecile sistem (standard sau specifice limbajelor de nivel înalt), ce pot fi incorporate în imaginile de task-uri ce formează o aplicație utilizator, pentru a efectua funcții complexe sau specifice, cum ar fi operațiile de intrare/ieșire ;

b) *Sistemul de gestiune a fișierelor* (**FCS/RMS**), asigurînd o interfață generală de acces a task-urilor de aplicație la mecanismul de intrare/ieșire.

Fiecare fel de servicii (**FCS** sau **RMS**) este compus din seturi de subrutine (în cod obiect), plasate în bibliotecile sistem standard, ce pot fi incorporate în imaginile de task-uri, pentru a efectua operații de intrare/ieșire și funcții de prelucrare a datelor. Serviciile **FCS** sau **RMS** sînt disponibile și sub forma unor biblioteci de rutine partajate, rezidente, la care task-urile utilizator se pot conecta, static sau dinamic. Utilizarea sistemului de gestiune a fișierelor asigură independența totală a programelor utilizator față de configurația reală de dispozitive de intrare/ieșire, specificarea dispozitivelor reale putîndu-se face, static, la editarea de legături, sau dinamic, la execuția programelor ;

c) *Prelucrarea directivelor de intrare/ieșire* **QIO** — reprezintă nivelul cel mai de jos al transferurilor de intrare/ieșire utilizator. Orice task poate lansa direct o cerere de intrare/ieșire utilizînd directiva Monitor **QIO**. Utilizarea **QIO** permite un control direct asupra dispozitivelor conectate la un sistem, pentru care există drivere.

d) *Tratarea intrării/ieșirii la nivel de Monitor*, se face prin :

— Task-uri ajutătoare de control **(ACP)** ;
— Componente Monitor :
  — Prelucrare **QIO** ;
  — Subrutine de intrare/ieșire ;
  — Drivere de intrare/ieșire.

Task-urile ajutătoare de control **(ACP)**, mențin structura și integritatea datelor memorate pe volume cu structură de fișiere. Ele asigură funcții relative la gestiunea cataloagelor și spațiului pe volume, translatînd cererile logice ale task-urilor utilizator în cereri simple adresate driverelor dispozitivelor cu structură de fișiere.

Serviciile de intrare/ieșire furnizate de Monitor constau din modulele de prelucrare ale directivei **QIO** și o colecție de subrutine utilizate de drivere pentru obținerea cererilor de intrare/ieșire din cozi, pentru tratarea întreruperilor, retururi de stare la completarea unei intrări/ieșiri etc. Aceste subrutine fac parte din Monitor și centralizează funcțiile comune ale driverelor, eliminînd secvențele multiple de cod. Cererile de intrare/ieșire **(QIO$)**, sînt plasate în cozile de intrare ale driverelor în ordinea priorității acestora (egală cu cea a task-ului emițător). După plasarea unei cereri în coadă, sistemul nu așteaptă terminarea acesteia, redînd controlul task-ului emițător. Dacă acesta nu poate continua execuția, așteptînd rezultatul transferului, task-ul trebuie să se autoblocheze prin emiterea unei directive sistem (**WTSE$**, sau cerere cu așteptare **QIOW$**). La terminarea transferului, informațiile de stare sînt plasate de driver într-un bloc de stare al intrării/ieșirii **(IOSB)** pe care-l poate testa utilizatorul.

9.2. Dispozitive de intrare/ieșire

Dispozitivele de intrare/ieșire suportate de sistemul **MIX** se împart în următoarele categorii :

— dispozitive de intrare/ieșire fizice ;
— dispozitive de intrare/ieșire logice ;
— pseudo-dispozitive de intrare/ieșire ;
— dispozitivul de intrare/ieșire nul.

Numărul și caracteristicile dispozitivelor de intrare/ieșire pot fi descrise la (re)configurarea sistemului. O parte din aceste caracteristici pot fi modificate dinamic, utilizînd interfața operator-sistem (comanda **MCL/DCL SET**). Fiecare dispozitiv are un *identificator unic*, format dintr-un *nume de dispozitiv* (2 caractere ASCII) și *un număr de unitate*, opțional, de 0 ÷ 3 cifre octale, urmate de două puncte (:). În cazul în care numărul de unitate este omis, sistemul consideră drept valoare implicită unitatea 0 (de exemplu, **MT:** indică unitatea de bandă magnetică zero). Numele de dispozitive, ce identifică și driverele de intrare/ieșire corespunzătoare, sînt rezervate în familia de sisteme de operare **MIX**. Pentru utilizatori sînt diponibile numai numele cuprinse în categoria ZA-ZZ.

9.2.1.. Dispozitive de intrare/ieșire fizice

Categoriile de dispozitive fizice recunoscute de familia de sisteme de operare **MIX** sînt :

— Terminale (**TT***nn* :), incluzînd :
  — teletype-uri de tipul ASR-KSR-33 și 35 ;
  — console de tip Centronics ;
  — teleimprimatoare ;
  — dispozitive de afișare alfanumerică DAF 1001, DAF 2015, DAF 2020, VDT 40C, VDT 132, VDT 135, VDT 125, VDT 240, ALFA-GRAF-200 ;
  — alte dispozitive compatibile.

  Aceste terminale pot fi conectate la următoarele interfețe de linie asincrone :
    — interfață 1 linie (tip DL11) ;
    — multiplexor de 8 linii (tip DZ11) ;
    — multiplexor de 16 linii (tip DH11) ;

— Benzi magnetice : de 800 bpi (**MT***nn*:) ; de 800/1 600 bpi (**MM***nn*:) ; de 1 600 bpi (**MS***nn*:) ; streaming (cu transfer continuu) (**MF***nn*:).

— Casetă magnetică dual (**CT***nn*:).

— Imprimante (**LP***nn* :), seriale sau paralele : de 300 linii/minut și de 800 linii/minut.

— Cititoare de cartele (**CR***nn*:).

— Cititor/perforator de bandă pe hîrtie (**PR***nn*:/**PP***nn*:).

— Cititor rapid de bandă pe hîrtie (**PR***nn*:).

— O mare varietate de discuri :
  — flexibile : simplă densitate (256 Ko) (**DX***nn*:) ; dublă densitate (512 Ko) (**DY***nn*:).
  — fixe : 512 Ko (**DF***nn*:) ; 1 Mo (**DS***nn*:).
  — amovibile : 20 Mo, 40 Mo, 50 Mo (**DP***nn*:) ; 50 Mo (**DD***nn*:) ; 100 Mo, 200 Mo (**DB***nn*:) ; 250 Mo, 500 Mo (**DR***nn*:).
  — încasetate : 2,5 Mo, 5 Mo (**DK***nn*:) ; 28 Mo (**DM***nn*:).
  — Winchester : 10 Mo (**DL***nn*:) ; 70 Mo, 200 Mo (**DU***nn*:).

— Bandă magnetică adresabilă (**DT***nn*:).

— Cuploare de comunicații asincrone și sincrone :
  — Interfața linie asincronă (**XL***nn*:) ;

— Interfață de linie paralelă (**XB***nn* :) ;
— Cuplor de comunicație sincron microprogramat tip DMC (**XM***nn* :).
— Interfețe de linie sincrone : între 2 ÷ 19 Kbauds (**XP***nn*:) ; între 2 ÷ 1 000 Kbauds (**XQ***nn*:) ; între 50 ÷ 9 600 bauds (**XW***nn* :).
— Cuplor de comunicație pentru rețele locale tip ETHERNET (**XE***nn*:) ;
— Preprocesor de comunicații microprogramat, tip KMC :
— cu multiplexoare asincrone tip DZ (COMM IOP-DZ) ;
— cu linii sincrone tip DUP (COMM IOP-DUP) ;
— cu imprimante (**LK***nn*:).
— Comutator de BUS (**BS***nn*:).
— Cuplor pentru realizarea de sisteme multi-procesor (**LR***nn*:, **LT***nn*:).
— Convertoare analog-numerice (**AD***nn*:, **AF***nn*:, **UD***nn*:).
— Subsisteme de control industrial (**IC***nn* :, **IS***nn*:).
— Subsisteme pentru calcule de laborator (**AR***nn*:, **LA***nn*:, **LS***nn*:).
— Subsisteme grafice (**GR***nn*:).
— Subsisteme de trasare (plottere, digitizoare, mese de trasare).
— Alte echipamente specializate sau nestandard.

Pentru toate aceste dispozitive, familia de sisteme de operare **MIX** pune la dispoziția utilizatorilor driverele de intrare/ieșire corespunzătoare.

9.2.2. **Dispozitive de intrare/ieșire logice**

Acestea constituie unul din mijloacele prin care task-urile își mențin independența față de configurația de intrare/ieșire reală, constituind o caracteristică de reconfigurare a sistemului.

Denumirea acestor dispozitive poate fi echivalentă cu cea a dispozitivelor standard (de exemplu, **CT03:**), sau poate fi aleasă la întîmplare (**XY:**) :

Asocierea dintre un *nume logic de dispozitiv* și o *unitate de dispozitiv fizic* se face prin comanda operator **MCL** de asignare (**ASN**).

Utilizarea numelor logice de dispozitiv în locul numelor fizice în task-urile utilizator generalizează posibilitățile de execuție, aplicația utilizator putîndu-se desfășura corect pe orice configurație de dispozitive fizice reală.

Atribuirile dispozitivelor logice la dispozitivele fizice pot fi :

— *locale*, aplicîndu-se numai la task-urile lansate în execuție de la terminalele pe care s-a executat comanda operator **ASN** ; atribuirile locale pot fi controlate de orice utilizator ;

— *globale*, aplicîndu-se tuturor task-urilor din sistem ; atribuirile globale pot fi controlate numai de utilizatorii privilegiați ;

— *de introducere în sistem*, pentru fiecare terminal în parte (la execuția comenzii **MCL HELlO** sau **DCL LOGin**), asociind pseudodispozitivul **SY :** unui dispozitiv utilizator (de regulă disc) specificat la crearea numărului de cont asociat. Utilizatorii privilegiați pot controla aceste atribuiri prin intermediul utilitarului de creare/actualizare numere de cont (**ACNT**) și prin comanda **ASSIGN/LOGIN**.

Atribuirile locale acoperă atribuirile globale conflictuale. Același nume de dispozitiv logic poate fi atribuit local unor dispozitive fizice diferite, de către diferiți utilizatori.

În operațiile de intrare/ieșire, numele logice au precedență față de numele fizice.

Sistemul de operare **MIX-PLUS** oferă un suport complet pentru *nume logice extinse*, cu care se pot face asignări (atribuiri) de *șiruri logice*. Spre deosebire de sistemul de operare **MIX**, în care un nume logic și șirul său echivalent erau limitate la forma **ddnn :**, în sistemul de operare **MIX-PLUS** se pot utiliza șiruri formate din 1 la 63 caractere alfanumerice, inclusiv caracterele speciale dolar (**$**) și concatenare (—) fără terminatorul final obișnuit (:).

Sistemul **MIX-PLUS** ține evidența perechilor nume logic-nume echivalent în următoarele patru tabele de nume logice :

— *tabela de nume logice asociate task-urilor* — conține intrările numelor logice locale unor task-uri distincte. În mod implicit, comenzile **MCL DEFine** și **ASN** plasează un nume logic în tabela de nume logice asociate task-urilor ;

— *tabela de nume logice asociate grupurilor de task-uri* — conține intrările numelor logice comune unor task-uri având același număr de grup în codurile de identificare utilizator (**UIC**-uri) ;

— *tabela de nume logice asociate sesiunilor utilizator* — conține intrările numelor logice locale utilizatorilor introduși în sistem (la execuția comenzii **MCL HELlO** sau **DCL LOGin**) ;

— *tabela de nume logice sistem* — conține intrări accesibile tuturor task-urilor din sistem.

Odată cu apariția numelor logice extinse, s-a schimbat și algoritmul prin care sistemul **MIX-PLUS** crează și translatează asignările logice. Anterior, și în sistemul de operare **MIX**, la crearea unui nume logic se forța translatarea imediată a asignării acestuia la dispozitivul fizic echivalent ; la utilizarea numelui logic nu mai aveau loc alte translatări. În prezent, la crearea unui nume logic, sistemul asignează acest nume cu numele echivalent (logic sau fizic), fără a fi însă translatat ; translatarea are loc numai la utilizarea numelui logic. La reasignarea unui nume logic, se reasignează toate numele logice asignate la acesta. Pentru a inhiba translatarea recursivă, se poate preciza elementul final prin calificatorul/**FINAL** la descrierea unei asignări logice.

Un nume logic extins poate identifica :

— *un dispozitiv fizic ;*

— *elementele sau totalitatea unui specificator de fișier.*

În cazul utilizării unui nume logic drept nume de dispozitiv, acesta trebuie terminat în mod obligatoriu cu caracterul (:).

La utilizarea formatului **ddnn:**, atît pentru numele logic cît și pentru șirul său echivalent, sistemul **MIX-PLUS** simulează comportamentul comenzii **ASN** existent sub **MIX**, în sensul că, la asignarea numelui logic, se forțează expandarea completă a șirului echivalent. Dacă șirul expandat specifică un dispozitiv de I/E existent, numele logic este asignat șirului nume de dispozitiv și translatarea este marcată drept finală. Dacă șirul echivalent este translatat în alt șir decît cel de identificare al unui dispozitiv, sistemul **MIX-PLUS** interpretează șirul drept nume logic extins și asignează șirul **ddnn:** numelui său echivalent. Translatarea finală va avea loc, în acest caz, la utilizarea numelui logic.

Monitorul **MIX-PLUS** pune la dispoziția utilizatorilor o serie de noi directive sistem pentru accesarea numelor logice :

— creare nume logice (**CLON$/CLOG$**) ;
— translatare nume logice (**TLON$/TLOG$**) ;
— translatare recursivă nume logice (**RLON$/RLOG$**) ;
— ștergere (eliminare) nume logice (**DLON$/DLOG$**) ;
— asignare număr logic unui dispozitiv descris de un nume logic (**ACHN$**).

Forma **XXXN$** a apelului de directivă este folosită pentru cazul unei identități depline între șirurile de descriere a numelor logice. Forma **XXXG$** a apelului de directivă este folosită pentru identificarea unor nume logice din care se elimină, în cazul unor nume logice de tip **ddnn:**, zerourile suplimentare din numărul de unitate și terminatorii de tip (:) (de exemplu, numele logice **DR005: :** și **DR5:** au aceeași funcționalitate).

9.2.3. Pseudo-dispozitive de intrare/ieșire

Un pseudo-dispozitiv de intrare/ieșire este un dispozitiv cu nume, ce nu corespunde nici unui dispozitiv fizic în sistem. Corespondența pseudo-dispozitive-dispozitive fizice este realizată prin operații de redirectare.

Existența unor astfel de pseudo-dispozitive este utilă în cadrul sistemului **MIX** în următoarele cazuri :

— pentru stabilirea unor dispozitive de intrare/ieșire implicite ;
— pentru eliminarea funcționării degradate a sistemului datorată căderii (nefuncționării) unor dispozitive fizice.

Pornind de la aceste considerente a fost generalizată utilizarea pseudo-dispozitivelor **MIX** pentru toate task-urile sistem și utilizator. În prezent task-urile ce se execută sub **MIX** comunică cu unul sau mai multe din următoarele pseudo-dispozitive : ieșire consolă (**CO:**) ; listing consolă (**CL:**) ; dispozitive de rețea (**NS:, NX:, HT:**) ; terminal de intrare/ieșire (**TI:**) ; dispozitiv suport pentru task-uri sistem și utilizator (**SY:**) ; dispozitiv suport pentru bibliotecile sistem (**LB:**) ; dispozitiv suport pentru fișierele de tipărire asincronă (spooling) (**SP:**) (numai **MIX/MIX-PLUS**) ; dispozitiv suport pentru fișiere temporare de lucru (**WK:**) (numai **MIX-PLUS**) ; terminal virtual (**VT:**) (numai **MIX-PLUS**).

În cazul utilizării acestor pseudo-dispozitive, transferul de informații se face către/de la dispozitivul fizic la care a fost redirectat pseudo-dispozitivul.

Alte task-uri sistem utilizează implicit pseudo-dispozitivele **TI:** (interfața operator, programe sistem) și **CO:** (task-ul de notificare a erorilor utilizator). Task-urile de dezvoltare de programe utilizează implicit pseudo-dispozitivele **SY:** și **LB:** pentru specificarea suporturilor de intrare sau ieșire (de obicei disc). Pseudo-dispozitivul **WK:** este utilizat implicit pentru precizarea suportului fișierelor temporare de lucru pentru task-urile de dezvoltare de programe (**MAC, TKB** etc.) și compilatoare.

Pseudo-dispozitivul **CO:** poate avea un driver asociat, pe sistemele de operare **MIX/MIX-PLUS**, ce interfațează subsistemul de înregistrare a mesa-

jelor operator (**COT**). În cazul în care acest subsistem nu este integrat (la configurare) sau nu este activat (prin comanda operator **SET/COLOG . . .**), pseudo-dispozitivul **CO:** este redirectat către consola operator **(TTO:)**.

Pseudo-dispozitivele de rețea **(NS:, NX;, HT:)** au drivere asociate ce implementează diferite niveluri arhitecturale ale software-ului de rețea **MININET/MIX(-PLUS)**.

Terminalele virtuale **(VT:)**, implementate numai pe sistemul **MIX-PLUS** printr-un driver specific, constituie un mijloc de comunicare și sincronizare între task-uri „tată" și „fiu", ce au stabilite relații de paternitate. Task-urile „fiu" utilizează terminalele virtuale drept terminale de intrare/ieșire **(TI:)** pentru transmiterea comenzilor și transferul datelor de intrare/ieșire.

Pseudo-dispozitivul **TI:** nu poate fi redirectat, avînd o utilizare particulară pentru fiecare task, sistem sau utilizator, în parte.

La inițializarea sistemului, pseudo-dispozitivele **SY:**, **LB:** și **SP:** (numai **MIX/MIX-PLUS**) sînt redirectate implicit către discul de încărcare. Pe sistemele **MIX/MIX-PLUS**, cu protecție multiutilizator, pseudo-dispozitivul **SY:** este asignat, la introducerea în sistem, dispozitivului precizat la crearea numărului de cont asociat utilizatorului.

Pseudo-dispozitivul **WK:**, pe sistemul de operare **MIX-PLUS**, este asignat (global), la inițializarea sistemului, către discul de încărcare.

9.2.4. **Dispozitivul de intrare/ieșire nul**

Acest dispozitiv, cu numele **NL:**, constituie un mecanism de test pentru task-urile utilizator sau sistem și nu un dispozitiv real, avînd următoarele funcțiuni :

— sursă infinită de date de intrare ;
— receptor nelimitat pentru datele de ieșire.

Pentru punerea la punct a programelor utilizator sau sistem ce necesită cantități mari de date de intrare și generează volume mari de date la ieșire, utilizarea dispozitivului de intrare/ieșire nul este oportună.

În acest caz e necesară numai o intervenție operator de redirectare sau reasignare a numărului logic atașat intrării sau ieșirii din programul utilizator către dispozitivul nul. În momentul execuției programului, dispozitivul nul returnează la intrare un cod de detecție sfîrșit de fișier **(IE.EOF)** și la ieșire un cod de efectuare cu succes a transferului **(IS.SUC)**.

9.2.5. **Dispozitive cu acces public sau individual**

În sistemele de operare **MIX/MIX-PLUS**, în care s-a selectat la generare opțiunea de protecție multiutilizator, se permite un control individual, de către utilizatori, asupra configurației de dispozitive de intrare/ieșire.

În acest caz, dispozitivele de intrare/ieșire se împart în următoarele categorii.

— *dispozitive cu acces public* — la care au acces toți utilizatorii din sistem. Definirea publică a unor dispozitive din sistem este permisă numai utilizatorilor privilegiați (comanda operator **SET/PUB**) ;

— *dispozitive cu acces individual* — la care are acces numai utilizatorul proprietar. Utilizatorii neprivilegiați își pot aloca dispozitive în acces individual utilizînd comanda operator **ALL**(*ocate*) (Alocare). Eliberarea dispozitivelor cu acces individual se face prin comanda operator **DEA**(*locate*) (Dealocare) ;

— *dispozitive fără proprietar (flotante)* — dispozitivele care nu sînt nici publice, nici particulare. Ele pot fi utilizate, inițializate etc., de către ambele categorii, fără ca sistemul să efectueze controale de acces.

Utilizarea particulară a unor dispozitive de către utilizatori este posibilă și recomandată în configurațiile mari, de timesharing, în care se doresc timpi scurți de răspuns sau există baze de date individuale, la care nu se dorește un acces exterior aplicației.

9.2.6. Atașarea/detașarea dispozitivelor de intrare/ieșire

Atașarea (Attach) este operația prin care un task (sistem sau utilizator) capătă un *control exclusiv* asupra unui dispozitiv de intrare/ieșire.

Operația este utilă cînd se dorește obținerea unor date de intrare sau rezultate într-un șir continuu, neîntrerupt, pe dispozitive de intrare/ieșire secvențiale, fără structură de fișiere, cum ar fi : terminalele, cititorul de cartele, imprimanta.

Dispozitivul atașat rămîne în controlul exclusiv al task-ului pînă în momentul detașării explicite a acestuia. Dacă alte task-uri din sistem solicită transferuri către un dispozitiv atașat, cererile corespunzătoare rămîn în așteptare în coada de intrare a driver-ului corespunzător, acesta prelucrînd numai cererile emise de task-ul care a solicitat atașarea.

Dacă execuția unui task se termină înainte de detașarea unui dispozitiv, operația este implicit efectuată de Monitor.

Detașarea (Detach) este operația prin care se pierde controlul exclusiv al unui task asupra unui dispozitiv de intrare/ieșire.

Dacă în momentul detașării coada de așteptare a driverului corespunzător conținea cereri în așteptare aparținînd altor task-uri, acestea sînt luate imediat în considerare.

9.2.7. Redirectarea și reasignarea dispozitivelor de intrare/ieșire

În cazul indisponibilității anumitor dispozitive de intrare/ieșire pe o configurație de lucru, un task ce efectuează transferuri de intrare/ieșire poate funcționa corect dacă se utilizează procedurile operator de redirectare (**RED***irect*) și reasignare (**REA***sign*).

Redirectarea este procedeul prin care se schimbă dispozitivul de intrare/ieșire indisponibil (neoperațional), cu un altul, fără a schimba atribuirile de numere logice. Această schimbare se realizează de fapt prin plasarea cererilor de transfer în coada driver-ului corespunzător noului dispozitiv.

Operația de redirectare nu afectează cererile de intrare/ieșire anterioare, plasate în coada driver-ului pentru dispozitivul indisponibil. Acestea vor rămîne blocate pînă la reintrarea în funcțiune a acestui dispozitiv sau pînă la terminarea (normală sau anormală) a task-ului utilizator (ce forțează eliminarea cererilor neonorate din cozi).

Redirectarea este efectuată utilizînd comanda operator **RED**(*irecf*).

Nu pot fi redirectate : terminalul de intrare/ieșire (**TI:**), un dispozitiv de intrare/ieșire atașat, un dispozitiv conținînd un volum montat logic.

Reasignarea este procedeul prin care se schimbă atribuirile numerelor logice existente deja în task. În acest caz, un număr logic utilizat pentru efectuarea unei operații de intrare/ieșire nu mai corespunde dispozitivului inițial (corespondență fixă la editarea de legături), ci altuia din cadrul celor disponibile la un moment dat.

Reasignarea este efectuată utilizînd comanda operator **REA**(*sign*).

Cele două proceduri, reasignarea și redirectarea, constituie elemente componente ale serviciilor sistem pentru asigurarea independenței programelor utilizator de o anumită configurație de dispozitive de intrare/ieșire.

9.3. Numere logice

Numerele logice (**LUN**) reprezintă interfața de comunicație între task-urile utilizator sau sistem și dispozitivele fizice de intrare/ieșire.

Fiecărui dispozitiv de intrare/ieșire i se poate asocia, static sau dinamic, un număr logic ; fiecare task din sistem își poate stabili o corespondență proprie între numerele logice și dispozitivele de intrare/ieșire fizice, corespondență ce se poate modifica în orice moment. Flexibilitatea acestei asociații contribuie puternic la asigurarea independenței programelor de dispozitivele de intrare/ieșire.

Numărul logic poate fi considerat o altă formă (mai scurtă) de identificare a unui dispozitiv periferic. Odată stabilită asocierea număr logic-dispozitiv fizic, aceasta permite o proiectare directă și eficientă a taskului utilizator la descrierea tabelară a dispozitivelor de intrare/ieșire.

Schimbarea asocierii număr logic — dispozitiv fizic (procedura operator de reasignare) duce la eliminarea din cozi a cererilor de intrare/ieșire corespunzătoare asocierii anterioare. Verificarea existenței unor cereri de intrare/ieșire necompletate (în așteptare) rămîne o sarcină a utilizatorului întrucît acesta decide schimbarea asocierii.

Pentru fiecare task instalat în sistem există o *tabelă de numere logice* (**LUT**), de lungime variabilă, plasată în antetul de task. Tabela conține un număr de intrări, specificat implicit (6) sau explicit (opțiunea **UNITS** la editarea de legături), fiecare avînd 2 cuvinte. O intrare corespunde unui număr logic și conține un pointer către dispozitivul fizic asociat curent la acel **LUN**. În momentul lansării unei cereri de intrare/ieșire, se specifică, implicit, un număr logic de apel ; sistemul identifică dispozitivul fizic asociat prin indexarea tabelei de numere logice cu numărul logic specificat.

Numărul de asocieri valide pentru un task este cuprins între 0 (fără tabelă de numere logice) și 255, dar nu poate depăși numărul de intrări specificat la editarea de legături.

Cuvîntul 1 al fiecărei intrări conține o informație de legare către blocul de descriere al unei unități de dispozitiv fizic (**UCB**) la care se asociază numărul logic corespunzător. Asocierea număr logic-dispozitiv fizic poate fi *indirectă* (utilizatorul poate forța referirea unei unități către o alta printr-o comandă operator de redirectare) sau *directă*.

Numerele logice nu au nici o semnificație pînă în momentul asocierii acestora cu dispozitive de intrare/ieșire fizice, asociere efectuată prin una din următoarele metode :

— la editarea de legături (opțiunea **ASG**) ;
— prin comanda operator **MCL** sau **DCL (REA)** ;
— dinamic, în timpul execuției task-ului (directiva sistem **ALUN$**).

În urma unei editări de legături implicite (fără utilizarea opțiunilor **UNITS** și/sau **ASG**), cele 6 intrări conținute în tabela **LUT** au următoarele atribuiri : **SY: (LUN1 ÷ LUN4)**, **TI: (LUN5)** și **CL: (LUN6)**.

9.4. Drivere de intrare/ieșire

În cadrul sistemului de intrare/ieșire **MIX**, driverele de intrare/ieșire îndeplinesc următoarele roluri :

— prelucrarea întreruperilor externe generate de dispozitivul atașat ;
— inițierea operațiilor de intrare/ieșire ;
— terminarea anormală a operațiilor de intrare/ieșire în curs de desfășurare ;
— efectuarea unor funcții specifice de dispozitiv :
  — la epuizarea unor intervale de timp (*„time-out“*) sau pentru recuperarea din avarie (*„power-fail“*) ;
  — la plasarea (trecerea) controlorului sau unității de dispozitive în starea on-line sau off-line (numai sub **MIX-PLUS**).

Driver-ul (sistem sau utilizator) constituie o parte integrantă a Monitorului **MIX** ; fiind un proces sistem specializat, el asigură servicii specifice pentru fiecare tip de dispozitiv de intrare/ieșire, degrevînd Monitorul de responsabilități dependente de dispozitiv. Driverul poate apela o serie de rutine de intrare/ieșire ale Monitorului și poate fi apelat de către acesta în unul din următoarele puncte de intrare : tratare întrerupere, inițiere intrare/ieșire, time-out dispozitiv, terminare intrare/ieșire, recuperare din avarie, schimbare de stare controlor de dispozitiv (numai **MIX-PLUS**), schimbare de stare unitate de dispozitiv (numai **MIX-PLUS**).

Driver-ul primește cererile de intrare/ieșire, plasate (FIFO) într-o coadă de intrare, prin intermediul Monitorului, după ce acesta a prelucrat cererile de intrare/ieșire (**QIO**). La terminarea operației de intrare/ieșire, driver-ul completează informațiile de stare corespunzătoare operației de transfer și le returnează monitorului pentru a fi transmise task-ului utilizator.

În efectuarea operației de intrare/ieșire, driverul utilizează o serie de structuri de date ce descriu interfața cu dispozitivul de intrare/ieșire și Monitor (**DCB**, **UCB**, **SCB**, **CTB** (numai **MIX-PLUS**), **KRB** (numai **MIX-PLUS**)).

9.5. Cereri de intrare/ieșire

Efectuarea unor operații de intrare/ieșire este solicitată prin intermediul unor cereri de intrare/ieșire.

Cererile de intrare/ieșire sînt servicii sistem apelate de task-uri (directivele Monitor **QIO$** și **QIOW$**), realizînd o multiplexare a utilizării dispozitivelor periferice între mai mulți utilizatori. Operațiile de intrare/ieșire au loc în paralel cu celelalte activități ale sistemului de operare.

Cererile de intrare/ieșire sînt transmise (prin mecanismul de întreruperi sincrone **SST** — cod **EMT 377**) către Monitor, la dispecerul directivelor **QIO**, ce determină acțiunea specifică serviciului solicitat. Acțiunea depinde de tipul cererii, de tipul dispozitivului, de existența unui task ajutător **ACP** asociat dispozitivului și de starea acestuia.

Cererile de intrare/ieșire pot fi de mai multe tipuri:

— *transfer și control* — executate de driverele specifice, ce efectuează serviciile solicitate;

— *acces la fișiere (acces logic)* — executate de task-urile ajutătoare de tip **ACP**. Dacă accesul la un task de tip **ACP** este permis pentru dispozitiv, acesta transformă cererea de acces la fișiere în mai multe directive **QIO** specifice, pe care le lansează direct către driverele implicate sau prelucrează cererea fără efectuarea de intrări/ieșiri. Task-urile de tip **ACP** asigură un acces logic, la dispozitivele de intrare/ieșire, și nu fizic, făcînd parte integrantă din mecanismul de asigurare a independenței de dispozitiv a programelor de aplicație;

— *fără operație* (**NOP**) — executate de rutinele centralizate din Monitor pentru a asigura independența de dispozitiv a task-urilor ce apelează funcții **FCS**.

După validarea și verificarea cererilor de transfer și control, Monitorul plasează aceste cereri, în ordinea priorității task-urilor solicitate, în cozile de intrare ale driver-elor de intrare/ieșire.

După plasarea în coadă a unei cereri de intrare/ieșire task-ul apelant nu așteaptă completarea operației (cu excepția cazului **QIOW$**). Sincronizarea task-ului cu terminarea operației de intrare/ieșire se poate face prin: autoblocare, întrerupere software asincronă (**AST**), sau teste periodice.

La terminarea unei operații de intrare/ieșire, se semnalizează task-ului informații de stare complete privind operația de transfer.

9.6. Task-uri ajutătoare de control

Task-urile ajutătoare de control (**ACP**-*Ancillary Control Processors*) reprezintă, în familia de sisteme de operare **MIX**, task-uri sistem privilegiate ce asigură o interfață de acces logic la anumite tipuri de dispozitive de intrare/ieșire (disc magnetic, bandă magnetică, dispozitive de comunicație).

Configurațiile tipice de sisteme de operare **MIX** conțin următoarele task-uri ajutătoare de control:

— *Procesor de control al volumelor disc* (**F11ACP**), ce asigură interfața cu dispozitive disc;

— *Procesor de control al volumelor de bandă magnetică* (**MTAACP, F11MSG**), ce asigură interfața cu dispozitive de bandă magnetică ;

— *Procesoare de control alc rețelei* **MININET/MIX (NETACP, NMVACP, RMTACP)**, ce asigură interfața logică de acces a unui utilizator la o serie de servicii ale rețelei **MININET/MIX**.

În cele ce urmează, se vor face precizări numai la task-urile **ACP** disc și bandă. În general, un asemenea **ACP** manipulează *containere* cu blocuri de informații. Aceste containere se numesc *fișiere*. Task-urile **ACP** tratează fiecare fișier ca un dispozitiv conținînd blocuri virtuale contigue (precizate de numere de bloc virtuale (**VBN**), numerotate de la 1 la *n*, unde *n* reprezintă ultimul bloc din fișier). Termenii de *bloc logic* și *virtual* descriu aceiași unitate fizică de memorare; numai ordinea de numerotare este diferită. Blocurile virtuale au asociate numere de bloc logic (**LBN**), dar blocurile logice nu au asociate numere de bloc virtuale (**VBN**) pînă la alocarea lor într-un fișier.

Virtual contigue nu înseamnă, în mod necesar și logic contigue. Un volum disc conține mai multe fișiere la un moment dat. Pe măsura umplerii unui disc, există zone contigue libere din ce în ce mai mici. Un task **ACP** creează sau extinde un fișier, plasînd porțiuni ale acestuia în diferite zone ale unui disc. Blocurile își mențin numerele virtuale, în secvență, dar nu mai sînt logic contigue.

Un task **ACP** independent de dispozitiv controlează structurile virtuale și logice aplicate informațiilor și translatează o structură în alta. De exemplu, dacă utilizatorul solicită acces la un bloc dintr-un fișier, **ACP**-ul asociat volumului ce conține fișierul calculează numărul de bloc logic ce corespunde numărului de bloc virtual specificat.

După calcularea numărului de bloc logic, task-ul **ACP** solicită accesul la bloc către driverul de I/E asociat dispozitivului ce conține fișierul. Driver-ul translatează această cerere într-o locație fizică de bloc, la care dispozitivul hardware efectuează o operație de citire sau scriere.

Task-urile **ACP** fac parte, de obicei, din mecanismul de prelucrare a directivei **QIO**. Ele pot fi activate sau dezactivate (inhibate) pentru un anumit dispozitiv. După activare, (de ex., montare), dispozitivul poate fi utilizat în contextul suportului logic pe care îl permite acel **ACP** pentru dispozitivul de I/E. Cererea de I/E nu este transmisă direct driverului, ci task-ului **ACP** asociat. Cererea logică recepționată de **ACP** poate fi :

— prelucrată complet de către task-ul **ACP**, fără efectuarea de intrări-ieșiri ;

— transformată într-o succesiune de cereri de I/E (directive **QIO**), emise de către task-ul **ACP** către driverul de I/E.

În realizarea funcțiilor de nivel logic efectuate de către un task **ACP**, acesta interacționează cu rutine Monitor generale sau speciale.

9.7. Sistemul de gestiune a fișierelor

Sistemul de gestiune a fișierelor, format din module de acces **FCS** (*File Control Services* — servicii de acces la fișiere) și/sau **RMS** (*Record Management Services* — servicii de gestiune a înregistrărilor), este plasat la nivelul superior al sistemului de intrare/ieșire **MIX** și permite accesul independent de dispozitiv la configurația de intrare/ieșire dată.

Utilizatorul solicită efectuarea de intrări/ieșiri la nivel logic sau virtual prin intermediul unui limbaj de comandă adecvat, de nivel superior.

Modulele de acces, ce implementează sistemul de gestiune a fișierelor, transformă aceste cereri în apeluri de servicii sistem (**QIO**).

Utilizarea Sistemului de gestiune a fișierelor **MIX** asigură independența totală a programelor de configurația de dispozitive de intrare/ieșire prin :

— limbaj superior de I/E, implementat prin macroinstrucțiuni specifice :

OPEN, CLOSE = acces la nivel de fișier ;

GET, PUT = acces la nivel de articol ;

READ, WRITE = acces la nivel de bloc.

— modalitate omogenă de tratare a dispozitivelor cu structură de fișiere și prelucrare secvențială a suporturilor cu acces direct (permițînd înlocuirea unui dispozitiv secvențial cu unul în acces direct și invers) ;

— accesarea logică a fișierelor la nivel de înregistrare, modulele de acces specifiee **FCS** sau **RMS**, efectuînd activitățile de buferare, înregistrare (păstrare/regăsire), blocare/deblocare și alte funcții de control asupra dispozitivelor de intrare/ieșire.

9.8. Nivele de transfer în sistemul de intrare/ieșire

În sistemul de operare **MIX** se lefinesc trei nivele posibile la care pot avea loc transferurile de intrare/ieșire :

a) *Nivel fizic*

Unitatea de transfer fizică este cea acceptată de către echipamentul hardware al fiecărui dispozitiv periferic (de exemplu, sectorul pentru disc).

b) *Nivel logic*

În acest caz, transferurile de intrare/ieșire se fac pe blocuri, de dimensiuni convenite cu sistemul de operare. Pentru cele mai multe dispozitive, transferul la nivel fizic este identic cu cel la nivel logic, blocurile logice corespunzînd direct blocurilor fizice.

De exemplu, un disc este împărțit în sectoare de 256 cuvinte, aceeași dimensiune cu cea a blocurilor logice pentru toate discurile suportate de familia de sisteme de operare **MIX**.

Pentru alte dispozitive, corespondența nu se poate face unu la unu. Discul cu capete fixe, de exemplu, este adresabil la nivel de cuvînt ; cu toate acestea nu au loc transferuri la nivel fizic cu acest dispozitiv. Informațiile sînt întotdeauna transferate în blocuri logice de 256 cuvinte.

Un alt exemplu îl constituie discul flexibil, pe care informațiile sînt înregistrate în sectoare fizice de 64 cuvinte fiecare. Blocurile logice sînt constituite din 4 blocuri fizice.

Pentru dispozitivele structurale pe blocuri (cum sînt, de exemplu, discurile) blocurile logice sînt numerotate începînd cu zero și pot fi adresate direct. Pentru dispozitivele nestructurate pe blocuri (de exemplu, terminalele), blocurile logice nu sînt adresabile.

c) *Nivel virtual*

La acest nivel, transferurile de intrare/ieșire au loc numai pentru fișiere deschise. În acest caz, sistemul de operare proiectează unu la unu, blocurile virtuale în blocuri logice. Pentru dispozitivele cu structură de fișiere (disc, bandă magnetică), blocurile virtuale au aceeași dimensiune cu blocurile logice și sînt numerotate începînd cu 1, fiind relative la începutul unui fișier, nu la dispozitivul de intrare/ieșire.

Pentru dispozitivele fără structură de fișiere, proiectarea blocurilor virtuale în blocuri logice este directă.

9.9. Interfețe de programare ale unei intrări/ieșiri

Mijloacele de programare ale unei intrări/ieșiri în cadrul sistemului de operare **MIX** sînt :

— servicii de intrare/ieșire Monitor, pentru o prelucrare directă, la nivelul cel mai simplu, a unei intrări/ieșiri ;

— servicii de intrare/ieșire **FCS** sau **RMS**, pentru prelucrări generale de fișiere și acces la nivel de înregistrare și bloc.

Tabela 9.1 rezumă interfețele de programare ale unei intrări/ieșiri în sistemul de operare **MIX**.

Tabela 9.1

Interfețe de programare ale unei I/E

Metodă	Interfață program	Componente I/E	Scop
1. I/E la nivel de înregistrare	Cereri **FCS** sau **RMS**	**OPEN, CLOSE, GET, PUT, ACP,** Drivere de I/E	— Structuri de fișiere pe disc magnetic, bandă magnetică. — O intrare/ieșire independentă de dispozitiv trebuie să utilizeze metode de acces la nivel de înregistrare
2. I/E la nivel de fișier	Cereri **OPEN, CLOSE** și cereri **QIO$ (QIOW$)**	**OPEN, CLOSE, ACP** (eventual), Drivere de I/E	— Structuri de fișiere pe disc magnetic, bandă magnetică, casetă magnetică, bandă perforată. — Utilizatorul implementează propriile sale metode de acces la nivel de înregistrare
3. I/E la nivel fizic	Cereri **QIO$ (QIOW$)**	Drivere de I/E	— Acces rapid la dispozitiv. — Structuri de fișiere incompatibile **MIX** (tratate de utilizator)

9.9.1. Servicii de intrare/ieşire ale Sistemului de Gestiune a Fişierelor SGF

Modulele de acces **FCS** şi **RMS** permit un acces independent de dispozitiv, la nivelul de fişier şi articol, pentru toate tipurile de dispozitive de /intrare/ieşire clasice. Cel mai general tip de acces permite programelor să prelucreze înregistrări logice, în care blocarea/deblocarea articolelor este efectuată automat.

Utilizatorii îşi pot defini şi structuri proprii de acces sau blocare/deblocare pe suporturi cu structură de fişiere, în scopul unui control particular al alocării zonelor tampon sau optimizării prelucrării unor înregistrări de format special.

Figura 9.2 prezintă interfeţele de programare cu sistemul de intrare/ieşire **MIX**.

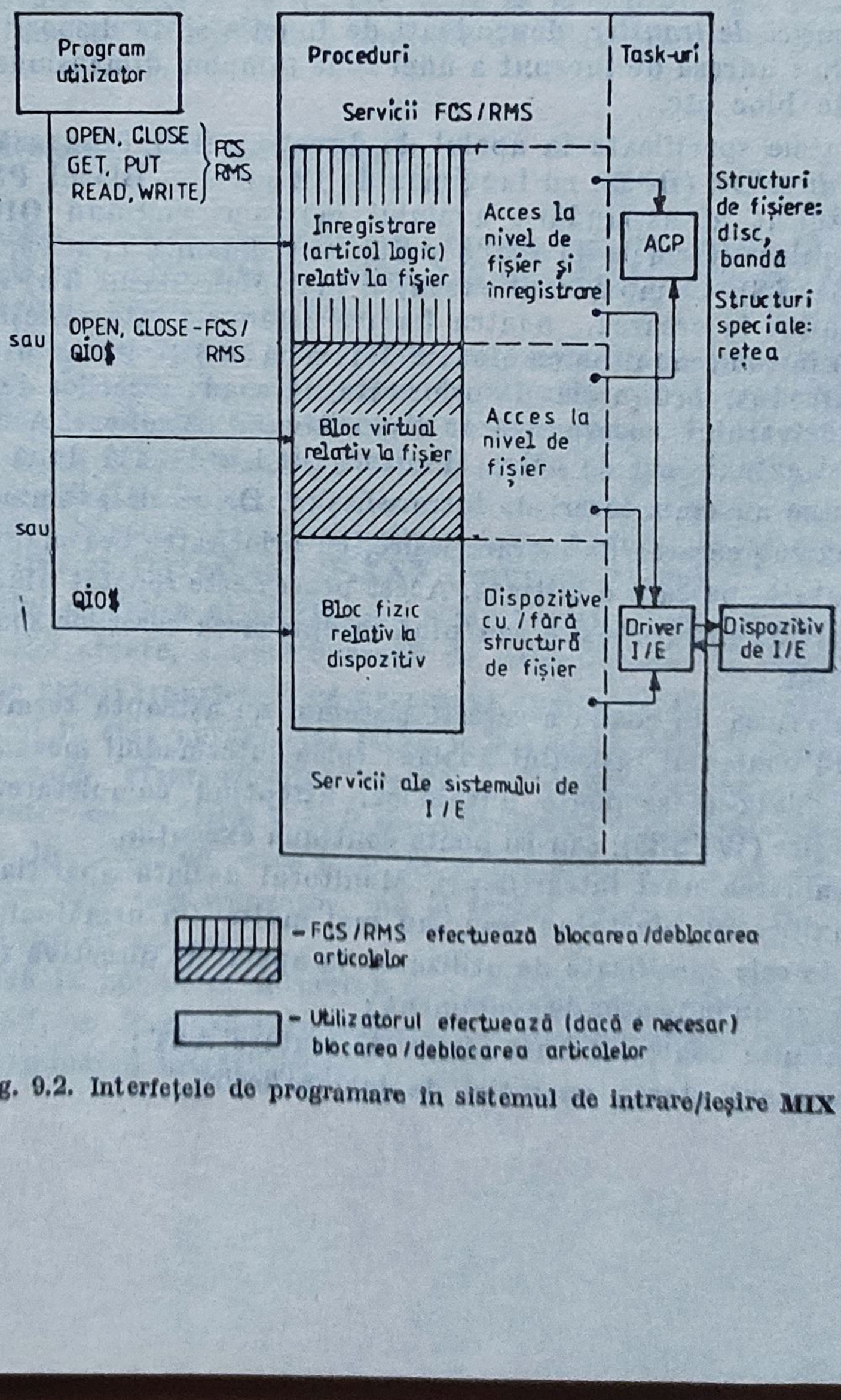


Fig. 9.2. Interfeţele de programare în sistemul de intrare/ieşire MIX

9.9.2. Servicii de intrare/ieşire Monitor

Cererile de intrare/ieşire sînt solicitate de către task-urile utilizator prin intermediul directivelor sistemului de intrare/ieşire (**QIO$** sau **QIOW$**).

La apelarea unei astfel de directive, task-ul utilizator specifică o serie de informaţii utilizate de Monitor pentru identificarea şi plasarea în coadă a cererilor de intrare/ieşire.

Informaţiile specificate sînt :

— *funcţia* de intrare/ieşire de efectuat ;

— *numărul logic* asociat dispozitivului pe care se desfăşoară operaţia de intrare/ieşire, şi opţional :

— *numărul indicatorului de eveniment* utilizat pentru sincronizarea execuţiei task-ului cu completarea operaţiei de intrare/ieşire ;

— *adresa unui bloc de stare* a intrării/ieşirii (**IOSB**), ce va conţine informaţii relative la modul de terminare a transferului ;

— *adresa unei rutine de tratare* **AST** asociată terminării operaţiei de intrare/ieşire ;

— *parametri de transfer*, dependenţi de funcţie şi de dispozitiv, cum ar fi, de exemplu : adresa de început a unei zone tampon, dimensiunea acesteia, un număr de bloc etc.

Argumentele specificate în apelul de directivă **QIO**, formează un *bloc de parametri ai directivei* (**DPB**), cu lungimea de 12 cuvinte. Blocul **PBD** este generat ca rezultat al expandării apelului macroinstrucţiunii **QIO$/QIOW$** *static*, la asamblare (formele de apel **$** şi **$C**), sau *dinamic*, în stivă, la execuţie (forma de apel **$S**). În momentul execuţiei task-ului, blocul **BDP** este utilizat de către Monitor la crearea, pentru fiecare cerere de intrare/ieşire, a unui pachet (**IOP**) în zona monitor cu alocare dinamică (**DSR**). Pachetul de intrare/ieşire este introdus, în funcţie de prioritate, în coada cererilor de intrare/ieşire ataşată driverului corespunzător dispozitivului specificat. Această coadă este creată şi actualizată de către Monitor, fiind ordonată după prioritatea task-urilor care au emis cereri de intrare/ieşire. Driverele examinează cozile asociate şi extrag cererea de intrare/ieşire, cu prioritatea cea mai mare, gata de a fi executată, pe care o iniţiază. Acest proces este repetat pînă la golirea cozii de cereri de intrare/ieşire sau pînă la epuizarea cererilor specificate de un task ataşat.

După plasarea în coadă a cererii, sistemul nu aşteaptă terminarea operaţiei, redînd controlul task-ului apelant (prin intermediul mecanismului de planificare). Task-ul se poate autobloca, aşteptînd completarea operaţiei de intrare/ieşire (**WTSE$**), sau îşi poate continua execuţia.

La terminarea unei intrări/ieşiri, Monitorul declară apariţia unui eveniment semnificativ, efectuînd una sau mai multe din următoarele operaţii (în funcţie de cele specificate de utilizator în apelul de directivă **QIO$**) :

— setează un indicator de eveniment ;
— transmite controlul unei rutine de tratare **AST** ;
— returnează starea operaţiei de intrare/ieşire.

9.9.3. Funcţii de intrare/ieşire standard

Gama operaţiilor de intrare/ieşire ce poate fi specificată prin intermediul directivei **QIO** este extrem de largă.

Există funcţii particulare pentru anumite dispozitive de intrare/ieşire, dar şi funcţii standard. Funcţiile standard sînt independente de dispozitiv şi pot fi solicitate pentru toată gama dispozitivelor de intrare/ieşire recunoscută de **MIX**.

Funcţiile standard **MIX** sînt :

— ataşare dispozitiv **(IO.ATT)** ;
— detaşare dispozitiv **(IO.DET)** ;
— anulare cereri de intrare/ieşire **(IO.KIL)** ;
— citire bloc logic **(IO.RLB)** ;
— citire bloc virtual **(IO.RVB)** ;
— scriere bloc logic **(IO.WLB)** ;
— scriere bloc virtual **(IO.WVB)**.

Specificarea funcţiei de intrare/ieşire se face printr-un nume de forma : **IO.XXX** (unde **XXX** identifică o operaţie de intrare/ieşire specificată).

9.9.4. Terminarea unei intrări/ieşiri

La terminarea unei intrări/ieşiri, cu succes sau cu eroare, Monitorul ia o serie de acţiuni dependente de parametrii specificaţi în apelul directivei **QIO.** Există *trei tipuri principale de retur* :

1. La terminarea unei operaţii de intrare/ieşire se declară un eveniment semnificativ. În cazul în care cererea de intrare/ieşire a specificat un număr de indicator de eveniment, evenimentul corespunzător este setat ;

2. În cazul în care s-a specificat, în apelul de directivă, un bloc de stare al intrării/ieşirii **(IOSB)**, octetul inferior (dreapta) al *primului cuvînd* din **IOSB** conţine codul de retur al intrării/ieşirii. Starea codului de retur este de forma **IS.XXX** (retur cu succes) sau **IE.XXX** (retur cu eroare).

Cuvîntul al doilea al **IOSB** este semnificativ numai în cazul terminării cu succes sau eroare, a unei operaţii de citire/scriere. El conţine de obicei contorul de octeţi transferaţi pe parcursul unei operaţii de intrare/ieşire.

În cazul în care parametrul **IOSB** a fost omis, task-ul apelant nu poate determina dacă intrarea/ieşirea cerută s-a desfăşurat normal. Returnarea unui indicator de condiţie **C = 0** după emiterea directivei semnifică numai acceptarea directivei de către Monitor şi plasarea cererii de intrare/ieşire în coada driverului corespunzător, nu şi faptul că operaţia de intrare/ieşire s-a desfăşurat cu succes ;

3. Dacă în apelul de directivă a fost specificată adresa unei rutine de tratare **AST**, se forţează transmiterea controlului către rutina de tratare **AST** la terminarea intrării/ieşirii.

9.9.5. Coduri de retur

La programarea unei intrări/ieșiri, Monitorul recunoaște și tratează două tipuri de condiții de stare :

— condiții asociate directivei **QIO**, ce indică acceptarea sau rejectarea acesteia ;

— condiții asociate intrării/ieșirii, ce indică execuția cu succes sau cu eroare a operației de intrare/ieșire.

Codurile de retur asociate tratării directivei **QIO** indică una din următoarele condiții :

— acceptarea directivei ;
— specificare de bufer (zonă tempon) invalid(ă) ;
— număr de indicator de eveniment invalid ;
— număr logic invalid ;
— indicator directivă sau dimensiune **DPB** invalide ;
— **LUN** neatribuit ;
— memorie dinamică insuficientă pentru alocarea **IOP** etc.

Codul de retur este returnat în cuvîntul de stare al directivei, identificat simbolic de locația **$DSW**, locație ce poate fi testată de programul utilizator.

Codurile de retur asociate intrării/ieșirii sînt poziționate de către Monitor și driver și returnate în blocul **IOSB**. Aceste coduri indică :

— terminarea cu succes a intrării/ieșirii ;
— operația de intrare/ieșire terminată anormal ;
— parametru **QIO** invalid ;
— dispozitiv neoperațional ;
— dispozitiv protejat la scriere ;
— încălcare privilegii ;
— fișier deja deschis ;
— detecție sfîrșit de fișier ;
— funcție ilegală ;
— eroare irecuperabilă ;
— dispozitiv off-line etc.

Octetul inferior al primului cuvînt al **IOSB** poate fi testat simbolic pentru determinarea tipului de cod de retur. Valorile binare ale codului de retur au următoarea semnificație :

— cod pozitiv mai mare ca zero = terminarea cu succes a operației de intrare/ieșire ;

— cod zero = operație în așteptare (prezentă în cozi sau lansată, dar neterminată) ;

— cod negativ = eroare de intrare/ieșire.

Există un număr de coduri de retur cu semnificație generală pentru toate dispozitivele de intrare/ieșire ; altele au o semnificație cu totul specială.

10

Drivere de intrare/ieşire MIX

10.1. Introducere

Un *driver de intrare/ieşire* este o componentă sistem sau utilizator ce asigură interfaţa dintre Monitor şi unul sau mai multe dispozitive de intrare/ieşire de acelaşi tip.

Rolul unui driver în sistemul de intrare/ieşire **MIX** este specific şi limitat, aceasta îndeplinind următoarele funcţiuni :

— recepţionează şi prelucrează întreruperile generate de dispozitivul(ele) de intrare/ieşire ;

— iniţiază operaţiile de intrare/ieşire solicitate driverului de către Monitor ;

— forţează terminarea anormală a unor intrări/ieşiri în curs de desfăşurare ;

— efectuează alte funcţii (specifice fiecărui dispozitiv), în cazul recuperărilor din avarie *(„power fail“)*, expirării unui interval de timp *(„time-out“)* sau trecerii controlorului/unităţii de dispozitiv în starea *„on-line“* sau *„off-line“* (numai sub **MIX-PLUS**).

Driverul iniţiază direct numai operaţiile de intrare/ieşire pe dispozitivul(ele) asociat(e) şi primeşte întreruperile generate de acest(e)a. Celelalte funcţii sînt executate ca urmare a unor apeluri explicite din Monitor. În general, driverul nu are o interacţiune directă cu task-urile de tip **ACP** ; cazuri particulare sînt driverele de reţea (ce interacţionează cu task-uri **ACP** de reţea) şi driverul de terminal (ce poate fi extins, pentru anumite tipuri de prelucrări şi terminale, cu o interfaţă de tip **ACP**, numită **ACD**).

Un driver de intrare/ieşire nu efectuează tratările specifice **QIO$**, acestea rămînînd în sarcina Monitorului **MIX**. Considerat a fi parte componentă a Monitorului, un driver posedă un context propriu, apelează rutine specifice sau generale Monitor, poate inhiba şi valida sistemul de întreruperi, putînd sincroniza de asemenea accesul la baza partajată de date sistem împreună cu alte componente ale Monitorului.

În cazul în care driverul suportă mai multe dispozitive de intrare/ieşire, fiecare avînd mai multe unităţi, toate putînd lucra în paralel, e necesară şi o sincronizare internă.

Driver ele utilizator trebuie să respecte convenţiile de programare ale sistemului **MIX**, pentru a păstra integritatea acestuia şi pentru a obţine performanţe deosebite de exploatare.

În familia de sisteme de operare **MIX** există cîte un driver pentru fiecare tip de dispozitiv de intrare/ieşire (fizic, logic sau nul). Un driver poate servi mai multe dispozitive de acelaşi tip, fiecare avînd ataşate mai multe unităţi.

Întrucît în implementarea sistemului de intrare/ieşire **MIX** s-a urmărit obţinerea unei eficienţe sporite (flexibilitate, economie de memorie, compatibilitate), driver-ele de intrare/ieşire nu sînt implementate ca task-uri, ci ca extensii ale nucleului sistemului de operare, putînd apela şi putînd fi apelate de către Monitor.

10.2. Locul driverelor în contextul nucleului sistem

Driver-ele constituie extensii logice ale Monitorului, apelate şi care apelează diferite servicii Monitor, pentru intrefaţarea dispozitivelor de intrare/ieşire. Pe sistemele cu relocare, driver-ul este mapat cu **APR**-ul 5, indiferent de dimensiunea zonei rezidente a Monitorului. Lungimea unui driver este restricţionată la 4 Kcuvinte (un **APR**) ; în cazul depăşirii acestei dimensiuni, driverul este direct răspunzător de maparea suplimentară cu **APR**-ul 6.

Monitorul asigură o serie de servicii relative la drivere ce pot fi de presau post- iniţiere. *Serviciile de preiniţiere* sînt efectuate de Monitor pe durata prelucrării directivei **QIO** :

— verificări de validitate asupra parametrilor **DPB** ;
— construirea pachetului de intrare/ieşire ;
— inserarea pachetului în coada cererilor de intrare/ieşire ;
— transferul controlului către driver la iniţierea unei intrări/ieşiri.

După iniţierea unei intrări/ieşiri, Monitorul apelează driver-ul pentru efectuarea altor funcţii specifice, privind :

— terminarea forţată a unor operaţii de transfer ;
— tratarea de time-out ;
— recuperarea din avarie.

Funcţiile de schimbare de stare controlor şi unitate de dispozitiv (existente numai sub **MIX-PLUS**), sînt exterioare operaţiilor de intrare/ieşire.

Serviciile de post-iniţiere sînt apelate de către driver după primirea de către acesta a controlului, la iniţiativa Monitorului sau în urma apariţiei unei (unor) întreruperi. Un astfel de serviciu important (rutina **$IODON**), îl reprezintă poziţionarea stării intrării/ieşirii către task-urile ce au solicitate operaţiile de intrare/ieşire.

Extragerea pachetelor **IOP** din coada cererilor de intrare/ieşire se face de către driver, utilizînd o rutină specifică Monitor (**$GTPKT/$GSPKT**).

Monitorul şi driverele interacţionează de asemenea în accesarea şi manipularea structurilor de date comune.

Accesul la aceste structuri partajabile este serializat prin intermediul unor *procese sistem (fork)*. După apelarea unei rutine Monitor specifice (**$FORK**),

driverul se transformă într-un proces sistem, plasat într-o coadă specifică de procese sistem *(fork)*. Monitorul salvează contextul driver-ului într-un bloc „fork" plasat în **SCB** și introduce (FIFO) acest bloc în coada proceselor sistem.

Prelucrarea proceselor *„fork"* se face în ordinea plasării lor în coadă. La redarea controlului, contextul driverului este restaurat și se reia execuția driver-ului cu prioritatea de întrerupere zero (secvența fiind complet întreruptibilă). Din acel moment, driverul are un acces complet și exclusiv la structurile de date Monitor.

Sub sistemul de operare **MIX-PLUS**, driverele utilizează acest mecanism și pentru accesarea structurilor sale de date. Un caz tipic, în acest sens, îl reprezintă efectuarea operațiilor de intrare/ieșire în paralel : fie între mai multe controloare de dispozitiv, fie între unitățile atașate aceluiași controlor. În acest caz utilizarea proceselor *„fork"* controlează accesul la structurile de date comune, iar coada proceselor *„fork"* reprezintă lista solicitărilor de acces.

Mecanismul proceselor *„fork"* este însă utilizat cu precădere în tratarea întreruperilor inițiate de dispozitivele de intrare/ieșire.

La apariția unei întreruperi, se dă controlul direct sau indirect (prin intermediul unui bloc de control al întreruperii — **ICB**), rutinei de tratare a întreruperii plasate în driver.

Prelucrarea întreruperilor în driver se efectuează *„mascat"* (la prioritatea 7). Acest mod neîntreruptibil nu poate fi menținut mai mult de 100 microsecunde pentru a nu se pierde alte evenimente externe, de timp real. În practică, tratarea la această prioritate este foarte scurtă în timp, trecîndu-se pe nivelul de prioritate al dispozitivului care a generat întreruperea. Tratarea pe acest nivel nu poate depăși însă 500 microsecunde. De cele mai multe ori, e nevoie de un timp mult mai lung de tratare. În aceste cazuri, prin intermediul listei de procese sistem, se efectuează o tratare secundară a întreruperii, pe o durată de timp nedeterminată.

Pentru eliminarea pierderilor de întreruperi în sistem, între Monitor și driverele de intrare/ieșire se stabilește un protocol strict de tratare a întreruperilor. La apariția unei întreruperi, se apelează o rutină de salvare a contextului întrerupr (**$INTSV/$INTSE**), direct de către driver (dacă acesta este rezident în zona Monitor) sau indirect, prin intermediul **ICB** (dacă driverul este încărcabil). Această rutină scade prioritatea de tratare de la nivelul 7 la cel al sursei de întrerupere. Ieșirea din tratarea de întrerupere se face prin rutina sistem **$INTXT**.

Pentru păstrarea integrității sistemului, se extinde tratarea de întrerupere prin apelarea rutinei Monitor **$FORK**, ce scade prioritatea de întrerupere la zero și asigură un acces controlat la structurile de date comune. Rutinele de tratare a întreruperilor se execută cu prioritate și pot fi întrerupte numai de cereri cu prioritate mai mare. Rutinele din lista *„fork"* se execută la momentul în care nu mai există întreruperi în așteptare și sînt complet întreruptibile. Task-urile se execută numai după golirea listei de procese *„fork"*.

Pentru driverele cu baze de date rezidente (al căror corp nu a fost încă încărcat în memorie prin comanda operator **LOA**), vectorii de întrerupere

asociaţi conţin intrări în rutinele Monitor de colcctare a întreruperilor parazite. Dacă subsistemul de înregistrare a erorilor este activat, întreruperile parazite se colectează în fişierul de erori sistem.

10.3. Structura unui driver de intrare/ieşire

Un driver de intrare/ieşire este structurat în două părţi mari : *instrucţiuni executabile* (codul sau corpul driver-ului) şi structuri ce formează *baza de date* asociată controloarelor şi unităţilor de dispozitiv suportate de către driver.

Corpul driverului este format dintr-o colecţie de rutine, fiecare acoperind o funcţie caracteristică. Pentru a putea fi apelate, de către Monitor, punctele de intrare asociate acestor funcţii sînt colectate într-o tabelă de dispecerare a driverului (**DDT**).

Fiecărui driver îi sînt asociate următoarele puncte de intrare :

a) *Tratarea de întrerupere* — este lansată atunci cînd un dispozitiv de intrare/ieşire, activat de driver, termină o operaţie de intrare/ieşire generînd o întrerupere către unitatea centrală. Legătura dispozitiv-driver este directă, prin intermediul vectorilor de întrerupere.

Adresa rutinei de tratare a întreruperii este plasată direct în vectorul de întrerupere (driver rezident) sau într-un bloc de control al întreruperii — **ICB** (driver încărcabil). Pentru driverele ce servesc dispozitive ce generează o singură întrerupere, punctul de intrare asociat este **$XXINT** ; pentru dispozitivele cu mai multe întreruperi (maxim 2 este permis), punctele de intrare asociate sînt **$XXINP** (tratare întrerupere recepţie) şi **$XXOUT** (tratare întrerupere emisie), cazul driverului de terminal şi a driverelor de comunicaţie.

Driverele din sistemul de operare **MIX-PLUS** acceptă mai multe întreruperi pentru fiecare dispozitiv. În acest caz, adresele punctelor de întrerupere sînt specificate în tabela de dispecerare **(DDT)** a driverului.

b) *Iniţierea unei intrări/ieşiri* — este activată de către Monitor atunci cînd există cereri de intrare/ieşire în coada de aşteptare a driverului, semnalizînd-i acetuia necesitatea prelucrării lor. Punctul de intrare asociat este ****INI**.

c) *Terminarea forţată a unor intrări/ieşiri în curs de desfăşurare* — este activată de către Monitor atunci cînd, în anumite împrejurări, e necesar ca driverul să termine o operaţie de intrare/ieşire, lansată, dar necompletată. Punctul de intrare asociat este *xx***CAN**.

d) *Tratarea de time-out* — la iniţierea unei operaţii de intrare/ieşire, driverul stabileşte un contor de numărare a timpului *(„time-out“)*. Dacă funcţia nu se termină în intervalul de timp specificat, Monitorul apelează driver ul pe această intrare pentru ca acesta să poată lua o decizie (reluarea operaţiei sau abandonarea ei). Punctul de intrare asociat este ****OUT**.

e) *Recuperarea din avarie* — este iniţiată de Monitor în trei cazuri distincte :

1) la restaurarea tensiunii de alimentare :

— după apariţia reală a unei căderi de tensiune, dacă controlerul de dispozitiv era marcat drept „ocupat“ (avea în desfăşurare o operaţie de intrare/ieşire) ;

— indiferent de starea operațiilor de intrare/ieșire, dacă s-a solicitat acest lucru în baza de date a driver-ului (bitul **UC.PWF**) ;

2) la inițializarea sistemului, simulîndu-se o restaurare a tensiunii de alimentare, permițînd driver-ului inițierea unor acțiuni specifice asupra bazei de date și a controlorului(elor) de dispozitiv(e) asociat(e) ;

3) la încărcarea în memorie a unui diver încărcabil :

— pe sistemul de operare **MIX**, dacă dispozitivul asociat era operațional și bitul **UC.PWF** setat ;

— pe sistemul de operare **MIX-PLUS**, se dă controlul în punctul de intrare **$XXLOA** (dacă a fost specificat), pentru efectuarea unor inițializări suplimentare.

În aceste cazuri se efectuează o reinițializare funcțională a driverului pentru reluarea (începerea) corectă a lucrului. Punctul de intrarea sociat este ****PWF.**

f) *Schimbarea de stare controlor* — este activată de către Monitor, sub sistemul de operare **MIX-PLUS**, ca urmare a unei solicitări de schimbare de stare a controlorului de dispozitiv, pentru reconfigurare : trecerea din starea on-line în off-line și invers. Punctul de intrare asociat este **$XXKRB.**

g) *Schimbarea de stare unitate* — este activată de către Monitor, sub sistemul de operare **MIX-PLUS**, ca urmare a unei solicitări de schimbare de stare a unei unități atașate unui controlor de dispozitiv, pentru reconfigurare : trecerea din starea on-line în off-line și invers. Punctul de intrare corespunzător este **$XXUCB.**

Punctele de intrare asociate schimbării de stare controlor și unitate sînt colectate în tabela de dispecerare a driverului **(DDT).**

Simbolii globali descriși anterior sînt verificați de către task-ul sistem **LOA**, prezența lor fiind obligatorie dacă driverul este încărcabil (cu o serie de excepții).

10.4. Structurile de date asociate unui driver

Structurile de date asociate unui driver de intrare/ieșire depind de numărul de controloare din sistem, de numărul de unități atașate la fiecare controlor și de caracteristicile suportate de respectivul dispozitiv de intrare/ieșire. Aceste structuri se complică odată cu creșterea numărului de caracteristici (facilități) suportate de driver. În general, un driver de intrare/ieșire interacționează cu următoarele structuri de date :

a) *Tabela de descriere a controloarelor de dispozitiv* **(CTB)** — există pentru fiecare controlor fizic din sistem (numai pentru driver-ele sistemului **MIX-PLUS**). Toate tabelele de descriere **CTB** sînt cuprinse într-o listă simplu înlănțuită, din zona de date sistem, capul de listă găsindu-se la locația **$CTBHD.** Fiecărui **CTB** îi este asociat un număr unic de controlor (2 caractere ASCII), ce permite Monitorului identificarea tipului de controlor (de ex., controlorul de tip RH11 are numele RH, deși la el se conectează dispozitivele cu numele DB, DS, DR sau MM).

Tabela **CTB** reprezintă o structură statică, ce conține informații generice de stare, legături și pointeri către alte structuri de date ale driver-ului **(KRB)** și care permite Monitorului dispecerarea întreruperilor asociate unui tip de controlor către driver-ele specifice, corespunzătoare dispozitivelor conectate la acel controlor ;

b) *Bloc(urile) de descriere a cererilor de acces la controloarele de dispozitiv* **(KRB)** — există pentru fiecare controlor din sistem (numai pentru driver-ele sistemului **MIX-PLUS**).

Blocurile **KRB** permit Monitorului păstrarea unor informații specifice privind interfața hardware și controlorul de dispozitiv și sînt folosite pentru accesarea corectă a informațiilor asociate unei unități conectate la un controlor (conținînd, printre altele, adresa vectorului de întrerupere, adresa **CSR** din pagina externă, informații de stare, masca de execuție pe **BUS**, etc.).

În configurațiile în care unitatea de dispozitiv are un acces singular la controlor și acesta nu admite mai multe operații de intrare/ieșire în paralel, blocul **KRB** este combinat (se suprapune) cu blocul de control al intrării/ieșirii **(SCB)** (ce păstrează contextul unității de dispozitiv pe durata operației de intrare/ieșire) (de ex., imprimanta, cititorul de cartele etc.).

În configurațiile în care sînt posibile mai multe operații de intrare/ieșire în paralel pe același controlor, contextul controlorului **(KRB)** este separat de contextul fiecărei unități de dispozitiv în parte **(SCB)**. În acest caz, Monitorul crează o coadă de cereri de acces la controlor în cadrul **KRB**, ce permite serializarea accesului la controlorul de dispozitiv. Pentru identificarea sursei de întrerupere în cazul unor operații paralele, blocul **KRB** conține o tabelă cu adresele blocurilor de control ale unităților de dispozitiv **(UCB)**, conectate la respectivul controlor ;

c) *Blocu(rile) de control al(e) unui dispozitiv de intrare/ieșire* **(DCB)** — există pentru fiecare tip de dispozitiv din sistem și descriu caracteristicile statice ale controlorului și unităților asociate acestora. Toate blocurile de control, **DCB** sînt cuprinse într-o listă simplu înlănțuită, din zona de date sistem, capul de listă găsindu-se la locația **$DCBHD**.

Majoritatea informațiilor din **DCB** sînt stabilite la asamblarea bazei de date a driverului și se referă la unitățile de dispozitiv, tipurile de funcții de I/E suportate de driver etc. Blocul **DCB** conține adresa tabelei de dispecerare a driveului **(DDT)**, adresa primului bloc de control al unității de dispozitiv **(UCB)**, fiind utilizat exclusiv de Monitor și nu de către driver.

În sistemul de operare **MIX-PLUS** pot exista mai multe blocuri **DCB** pentru același tip de dispozitiv ;

d) *Bloc(urile) de control al(e) unităților de dispozitiv* **(UCB)** — există pentru fiecare unitate de dispozitiv, conținînd caracteristicile individuale ale acesteia și o serie de parametri dinamici. Informațiile statice de descriere a unităților de dispozitiv sînt stabilite la asamblarea bazei de date a driver-ului iar cele dinamice pot fi modificate prin comandă operator **(SET)**, de către Monitor sau de către driver.

Blocul **UCB** conține pointeri către celelalte structuri de date asociate dispozitivului de intrare/ieșire **(DCB, SCB)**, informații de control asociate unității de dispozitiv (folosite la prelucrarea directivei **QIO**), definiții de stare ce descriu condiții operaționale, definiții de caracteristici dependente de

dispozitiv **şi** unitate, zone de memorare. În general, blocurile **UCB** conţin o zonă **standard**, cu o structură identică pentru toate tipurile de dispozitive, şi o zonă specifică pentru fiecare tip ;

e) *Bloc(urile) de control al(e) stării intrării/ieşirii* (**SCB**) — conţin contextul driver-ului pentru operaţiile de intrare/ieşire desfăşurate pe o unitate de dispozitiv. Există cîte un bloc **SCB** pentru fiecare controler de dispozitiv (**DCB**), dacă acesta nu admite operaţii de intrare/ieşire în paralel, sau pentru fiecare unitate conectată (**UCB**) dacă controlerul admite operaţii în paralel (numai în sistemul MIX-PLUS).

În sistemul de operare **MIX-PLUS**, pentru operaţiile care nu se desfăşoară în paralel, blocul **KRB** este suprapus peste blocul **SCB**. Pentru operaţiile care se desfăşoară în paralel, blocul **SCB** este definit şi conţine un pointer către blocul **KRB** asociat controlorului la care este conectată unitatea de dispozitiv.

Blocul **SCB** conţine informaţii privind lista pachetelor de intrare/ieşire, lista blocurilor „*fork*“, contextul procesului sistem („*fork*“), informaţii de stare şi de eroare. Aceste informarii sînt stabilite în general la asamblarea bazei de date a driverului.

Monitorul accesează blocul **SCB** la iniţierea unei operaţii de intrare/ieşire, la salvarea contextului unei operaţii în curs de desfăşurare şi pentru setarea stării intrării/ieşirii. Driver-ul are acces, în general, la blocul **SCB** numai în citire ;

f) *Tabela de dispecerare a driverului* (**DDT**), conţine puncte de intrare în driver (apelate de către Monitor) şi adrese ale rutinelor de tratare a întreruperilor (numai sub sistemul **MIX-PLUS**). Tabela **DDT** este identificată de simbolul global **$XXTBL**, unde **XX** reprezintă mnemonica de identificare a dispozitivului de intrare/ieşire (două caractere ASCII). Tabela de dispecerare trebuie să se găsească în primele 4 Kcuvinte din corpul driver-ului (primul **APR**) ; de obicei, această tabelă este plasată înainte de codul executabil propriu-zis. Dacă lungimea driver-ului depăşeşte 4 kcuvinte, punctele de intrare trebuie să se găsească în primul **APR**, iar driverul este răspunzător de maparea suplimentară. Corpul driverului nu poate fi segmentat. Generarea tabelei de dispecerare se face utilizînd macroinstrucţiunea **DDT$**.

Relaţiile între structurile de date, prezentate mai sus, diferă în funcţie de tipul de dispozitiv de intrare/ieşire şi sistemul de operare.

Fig. 10.1 reprezintă structurile de date pentru un controlor de imprimantă (LP), cu o unitate ; sub sistemul de operare **MIX**, există un **DCB**, un **UCB** şi un **SCB**. Sub sistemul **MIX-PLUS**, există un **CTB**, un **DCE**, un **UCB** un **SCB** şi un **KRB** care se suprapune parţial peste **SCB**.

Fig. 10.2 reprezintă structurile de date pentru un controlor de bandă magnetică (tip **MM**), cu 2 unităţi ; sub sistemul de operare **MIX**, există un **DCB**, două **UCB**-uri şi un singur **SCB**, întrucît numai una din cele două unităţi poate fi activă la un moment dat. Sub sistemul de operare **MIX-PLUS**, există un **CTB (RH)** comun pentru mai multe tipuri de dispozitive (**DB**, **DS**, **DR** şi **MM**), un **KRB** specific (**RHD**), două **UCB**-uri şi două **SCB**-uri, întrucît cele două unităţi pot efectua operaţii de intrare/ieşire în paralel.

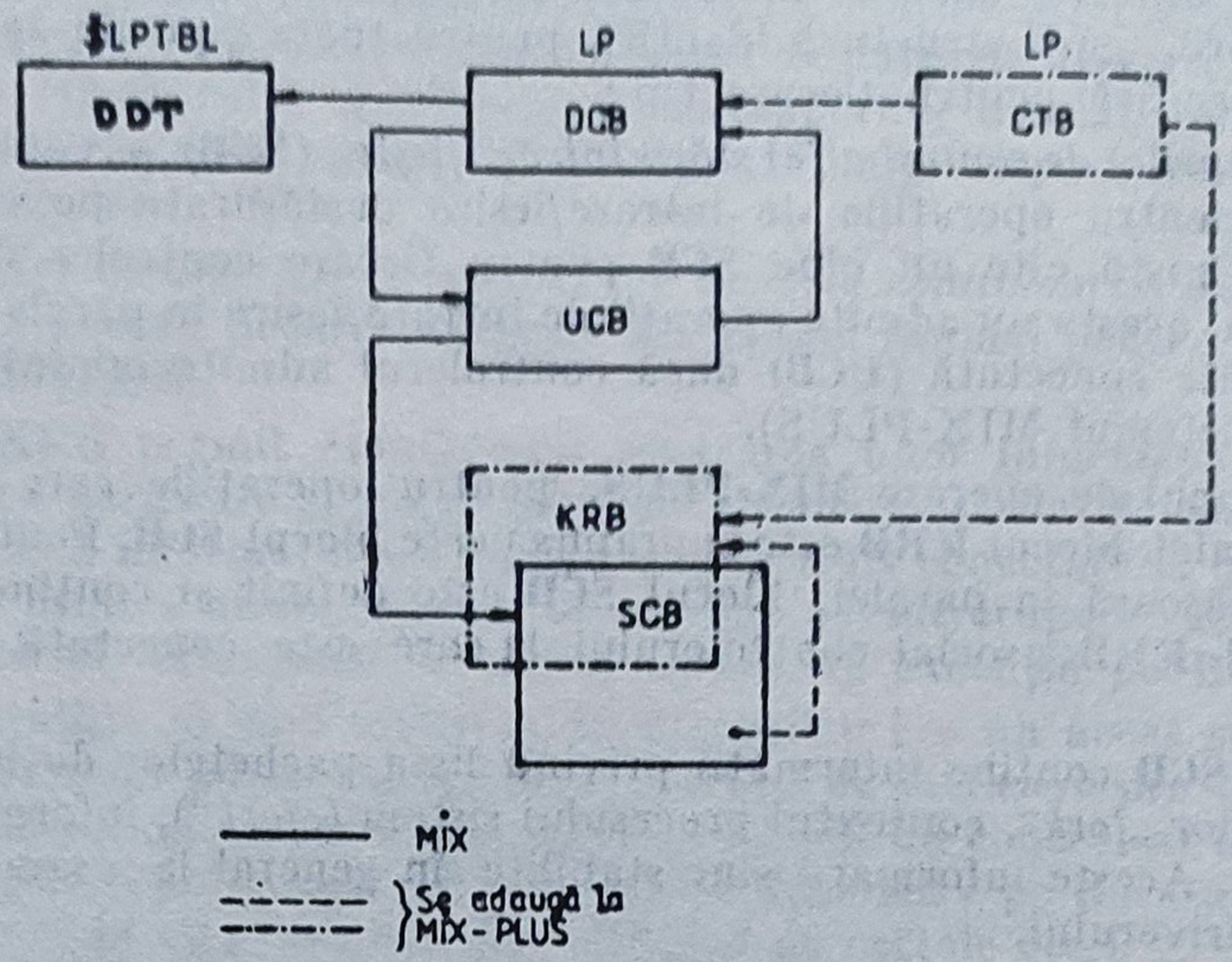


Fig. 10.1. Structuri de date pentru un controlor de imprimantă cu o unitate, sub sistemele MIX și MIX-PLUS

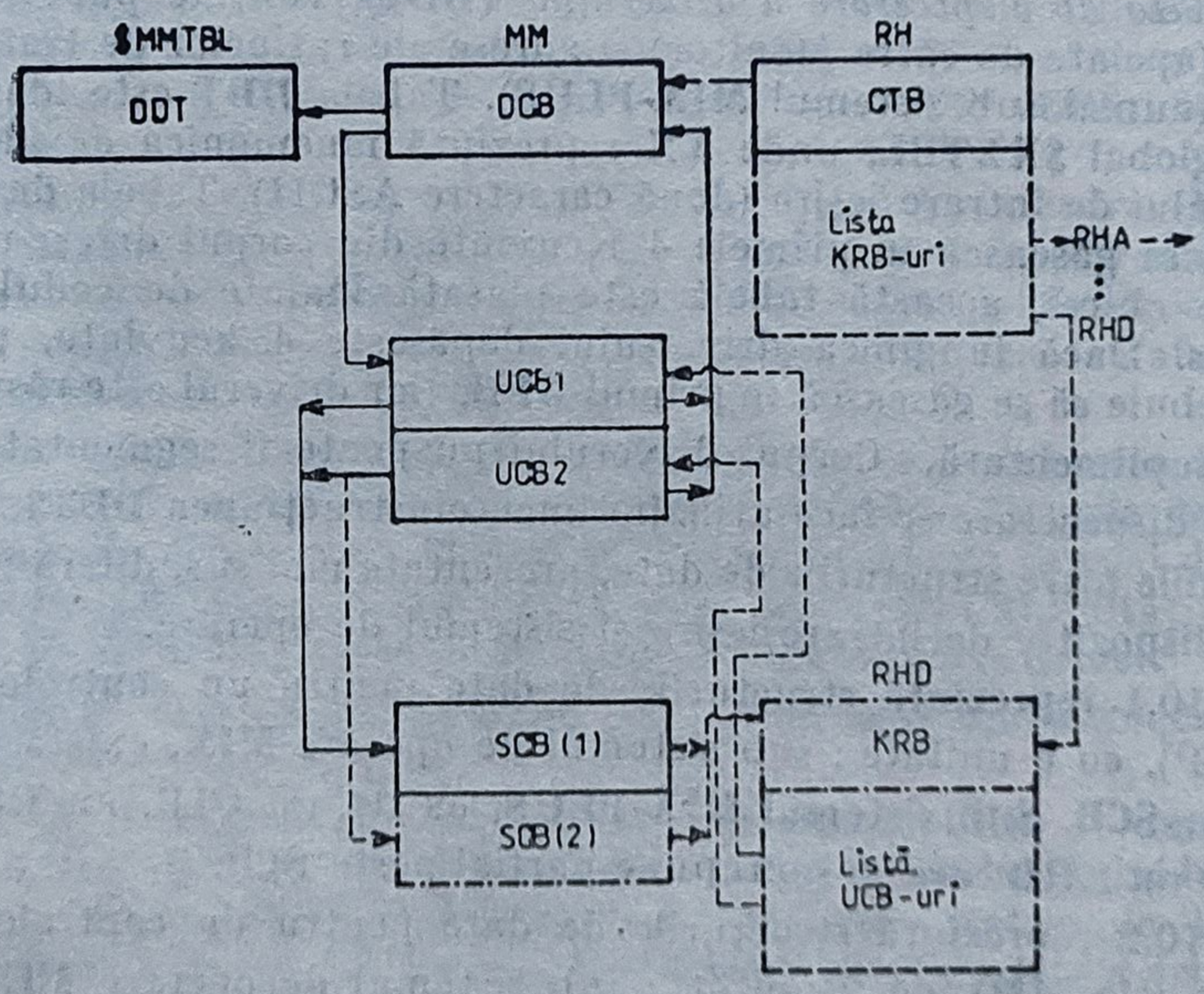


Fig. 10.2. Structuri de date pentru un controlor de bandă magnetică cu două unități, sub sistemele MIX și MIX-PLUS

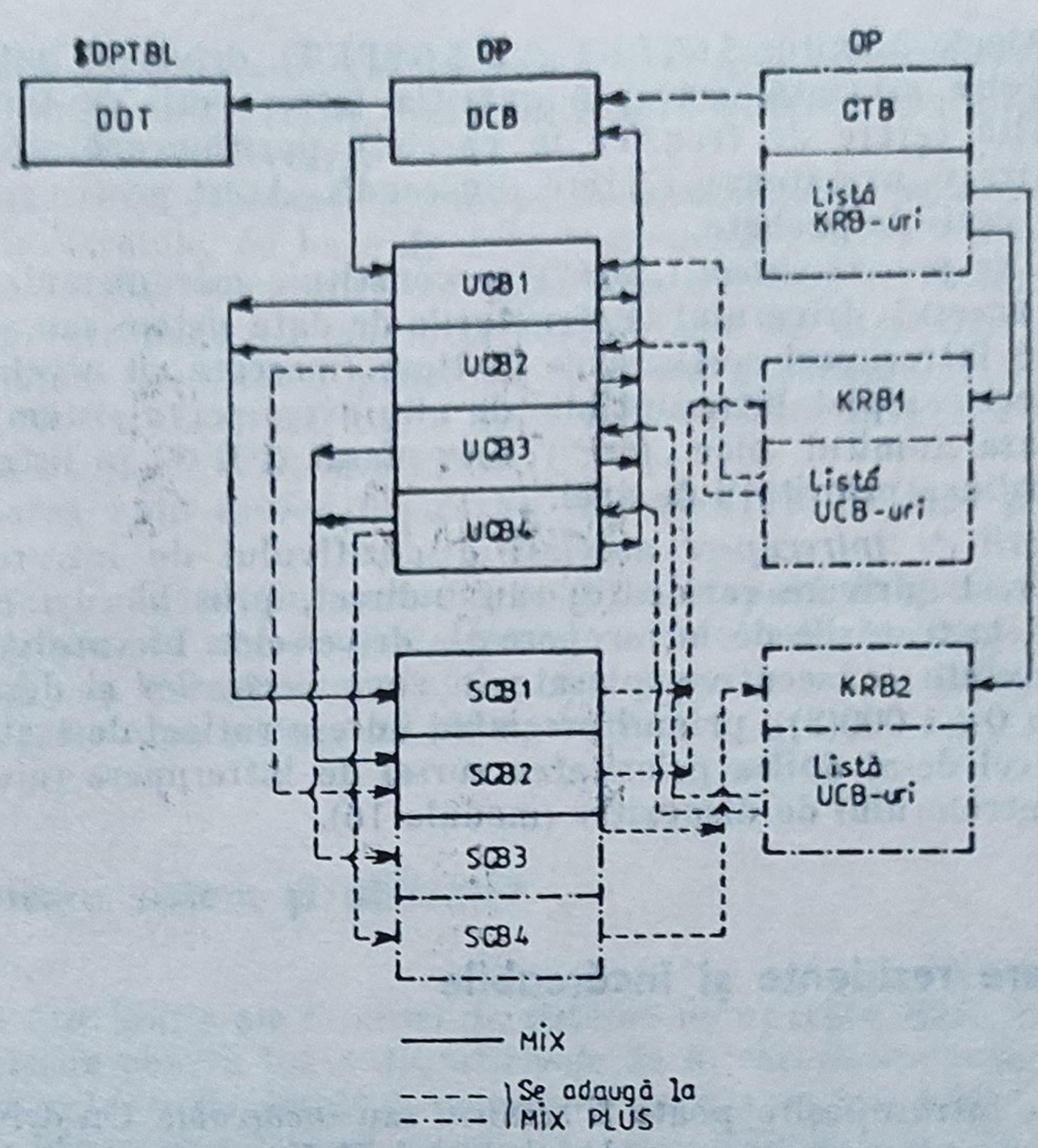


Fig. 10.3. Structuri de date pentru două controloare de disc, fiecare cu două unități, sub sistemele MIX și MIX-PLUS

Fig. 10.3 reprezintă structurile de date pentru două controloare de disc (tip **DP**), fiecare cu 2 unități ; sub **MIX** există un **DCB**, patru **UCB**-uri și 2 **SCB**-uri, întrucît cele două controloare pot opera în paralel. Sub sistemul de operare **MIX-PLUS**, există un **CTB**, un **DCB**, două **KRB**-uri, patru **UCB**-uri și patru **SCB**-uri, întrucît cele patru unități pot efectua operații de poziționare capete de citire/scriere („*seek*") în paralel.

g) *Pachetul de intrare/ieșire* **(IOP)** — conține informații extrase din cererea de intrare/ieșire (blocul **DPB** folosit de directivele **QIO$/QIOW$**). La lansarea unei cereri de intrare/ieșire, Monitorul efectuează o serie de verificări de validitate asupra parametrilor din **DPB**. Dacă aceste verificări se termină cu succes, Monitorul alocă un pachet de intrare/ieșire **(IOP)** din zona sistem cu alocare dinamică **(DSR)** și îl inserează în coada pachetelor de intrare/ieșire, specifică dispozitivului.

h) *Coada cererilor de intrare/ieșire* — conține pachetele de intrare/ieșire **(IOP)** ce urmează a fi prelucrate de driver.

Fiecărui controlor sau unitate de dispozitiv îi corespunde o coadă de cereri de intrare/ieșire, ordonată în funcție de prioritatea task-urilor emițătoare. Capul de listă (simplu înlănțuită), asociat fiecărei cozi, este plasat în **SCB**. Extragerea pachetelor din coadă se face la inițiativa driverului, în secvența de inițiere a unei intrări/ieșiri. După extragerea unui pachet **IOP**

(folosind rutinele Monitor **$GTPKT** sau **$GSPKT**), driver-ul inițiază funcția de intrare/ieșire asociată, așteaptă apariția întreruperii de terminare (sau inițiază o altă cerere de transfer în paralel), poziționează starea intrării/ieșirii și extrage următoarea cerere din coadă. Acest proces continuă pînă la epuizarea cozii de pachete.

i) *Lista de procese sistem* („*fork*") — constituie mecanismul prin care se serializează accesul driverului la structurile de date sistem sau se efectuează prelucrări de întreruperi costisitoare ca timp (mascate cu nivelul de prioritate zero, deci complet întreruptibile de alte evenimente sistem). Contextul driverului (așa numitul bloc „*fork*") este plasat (FIFO) în lista de procese sistem, în ordinea priorității de apel.

j) *Vectorii de întrerupere* asociați dispozitivului de intrare/ieșire sînt conectați direct (drivere rezidente) sau indirect, prin blocuri **ICB** (drivere nerezidente), la tratările de întrerupere ale driver-elor. Fiecare vector constă din două cuvinte consecutive, plasate în zona vectorilor și derutărilor din sistem (zona 0÷1 000(8)), primul precizînd adresa rutinei de tratare a întreruperii, iar cel de-al doilea prioritatea sursei de întrerupere (nivelul **BR**) și numărul controlorului de dispozitiv (modulo 16).

10.5. Drivere rezidente și încărcabile

Un driver de intrare/ieșire poate fi *rezident* sau *încărcabil*. Un driver rezident constituie o parte componentă a Monitorului **MIX**, încorporată la generarea sistemului. Un driver nerezident constituie o extensie a Monitorului **MIX** ce poate fi încărcată sau eliminată (descărcată) din memorie dinamic.

Alegerea uneia sau alteia din soluții este condiționată de memoria disponibilă la un moment dat și de frecvența transferurilor de intrare/ieșire pe dispozitivele periferice. În cazul în care volumul de intrări/ieșiri prelucrate de un număr de drivere (sistem sau utilizator) este mare, se recomandă soluția rezidenței acestor drivere în zona Monitor. Soluția are dezavantajul creșterii dimensiunii Monitorului în detrimentul spațiului afectat utilizatorilor.

În cazul în care restricțiile de memorie devin drastice, se recomandă soluția driver-elor încărcabile. În acest caz va rezulta un Monitor mai mic și mai mult spațiu pentru zona sistem cu alocare dinamică (**DSR**).

Pentru a micșora zona Monitor se recomandă construirea unor drivere încărcabile (sistem sau utilizator) cu structură de date și cod încărcabile.

Aducerea în memorie a unui driver încărcabil se face printr-o simplă comandă operator — **LOAd**, implementată sub forma unui task sistem. Acesta efectuează o serie de verificări asupra bazei de date asociate driver-ului, relocă pointeri în spațiul de adresare al Monitorului și încarcă codul și baza de date a driverului.

În procesul de încărcare a unui driver se construiește, în zona cu alocare dinamică a Monitorului, baza de date a driverului (dacă aceasta nu era deja rezidentă în memorie), se conectează porțiunea de cod la vectorul de întrerupere corespunzător și se încarcă porțiunea de cod a driverului într-o partiție utilizator sau sistem. În cazul sistemelor cu relocare, legătura între vectorii

de întrerupere și partiția de rezidență a driver-ului este realizată de Blocul de control al întreruperii (**ICB**), cîte unul pentru fiecare tratare de întrerupere distinctă efectuată în driver.

Toate driverele suportate standard de sistemele de operare **MIX/MIX-PLUS** sînt încărcabile, cu baze de date încărcabile (inclusiv driver-ul de terminal). Această soluție permite o flexibilitate extremă în configurarea sistemului de operare pentru o anumită configurație de dispozitive de intrare/ieșire, eliminînd necesitatea generării repetate a sistemului pentru fiecare nouă instalare. În acest caz, toate driverele utilizator urmează a se construi cu structură de date și cod încărcabile.

Eliminarea unui driver încărcabil din memorie se face prin comanda **UNL***oad*.

În urma eliminării din memorie, partiția ocupată de driver se eliberează putînd fi utilizată în alte scopuri. Procedura de eliminare șterge baza de date a driverului, cu excepția cazului în care aceasta a fost integrată în zona Monitor la generarea sistemului de operare.

10.6. Drivere sistem și utilizator

Seturile de distribuție ale familiei de sisteme de operare **MIX**, includ drivere de intrare/ieșire pentru toate dispozitivele de intrare/ieșire standard, prezentate în paragraful corespunzător Sistemului de intrare/ieșire **MIX**.

Aceste drivere sînt drivere sistem și fac parte din sistemul de operare **MIX**.

În cazul în care un utilizator dorește să conecteze la configurația sa hardware un dispozitiv de intrare/ieșire pentru care nu există un driver asociat, e necesară scrierea acestuia de către utilizator.

Sistemul de operare **MIX** permite și pune la dispoziția utilizatorilor mijloacele necesare scrierii și incorporării unor noi drivere utilizator. Un asemenea exemplu de scriere a unui driver utilizator este tratat în cap. 38 (vol. 2).

10.7. Drivere vectorizate

În scopul asigurării independenței unor drivere (sistem sau utilizator) față de varianta de sistem existentă, în versiunea de sistem **MIX-PLUS V2.0** se utilizează pe larg facilitatea de vectorizare a driverelor (pricipiile vectorizării sînt prezentate pe larg în cap. 5, paragraful 11).

Spre deosebire de task-urile privilegiate, driverele nu pot folosi directive **GIN$** pentru translatarea vectorului local. Pentru translatare se apelează o rutină, executabilă în stare sistem, identificată indirect prin intermediul unei locații de memorie fixe (112_8 în spațiul virtual/fizic al driver-ului/memorie), ce conține adresa unei tabele de mapare a comunului Monitor ce include rutina de vectorizare.

Secvența de apel a acestei rutine, în cadrul unui driver, poate fi plasată în modulul de inițializare a unei operații de I/E sau în secvențe de tratare a recuperării din avarie (cădere de tensiune).

Vectorizarea driverelor de I/E (folosită pe larg în realizarea majorității driverelor sistem) permite folosirea acestora, fără modificări, pe diversele variante ale sistemului de operare **MIX-PLUS V2.0** (cu sau fără spații **I** și **D**), precum și în cazul unor versiuni ulterioare de sistem.

10.8. Facilități avansate oferite de driverele sistem

Driverele sistem, existente sub sistemele de operare **MIX/MIX-PLUS**, conțin o serie de facilități avansate pentru creșterea gradului de paralelism între operațiile de intrare/ieșire, optimizarea accesului la volumele disc, partajarea unităților de dispozitiv, economia de memorie și creșterea performanțelor generale de exploatare.

Aceste facilități, implementate în driver-ele sistem, sînt asistate de rutine și servicii specifice Monitor :

a) *Efectuarea de operații de intrare/ieșire full-duplex*

În mod normal, un driver prelucrează un singur pachet de intrare/ieșire **(IOP)** la un moment dat. La preluarea pachetului, unitatea de dispozitiv este marcată ca ocupată și nu se mai admit operații de intrare/ieșire suplimentare pînă la terminarea primei operații de transfer.

Operarea full-duplex permite efectuarea mai multor intrări/ieșiri pe același dispozitiv, în același interval de timp (de ex., driverul de terminal permite efectuarea unei recepții și transmisii de date în paralel, iar driverul subsistemului pentru aplicații de laborator permite efectuarea a maximum opt operații de intrare/ieșire simultan).

În acest caz este necesară extinderea structurilor de date **UCB** și **SCB** pentru păstrarea contextului integral al operațiilor de intrare/ieșire desfășurate în paralel, fiind necesară și o sincronizare internă a accesului la aceste structuri de date, prin intermediul proceselor sistem.

b) *Efectuarea de operații de intrare/ieșire buferate*

De obicei transferurile de intrare/ieșire au loc în zona de memorie atașată task-ului care le-a inițiat. Pe durata transferului de date, task-ul nu poate fi evacuat, blocînd resursa memorie internă.

Pentru eliminarea acestei situații, driverele pot apela rutine sistem ce asigură stocarea datelor în memoria alocată driverelor sau zona Monitor cu alocare dinamică **(DSR)**, permițînd evacuarea task-urilor pe durata transferurilor de intrare/ieșire (un exemplu tipic este reprezentat de driver-ul de terminal). La terminarea transferului de intrare/ieșire buferat, task-ul evacuat este readus în memorie și datele transferate (în buferul asociat driver-ului sau **DSR**) sînt copiate în buferul utilizator (în zona asociată task-ului).

c) *Optimizarea accesului la coada cererilor de intrare/ieșire*

Coada cererilor de intrare/ieșire **(IOP)** este organizată FIFO, pe baza priorității task-urilor ce au solicitat efectuarea operațiilor de intrare/ieșire.

Pentru creşterea performanţelor de tansfer la volumele disc, sistemul de operare **MIX-PLUS** asigură un set de algoritmi de optimizare a accesului la coada cererilor de intrare/ieşire pentru cererile de prioritate egală sau cuprinse într-o gamă de priorităţi, specificată static (la generarea driver-ului) sau dinamic (comanda operator **SET**) de către inginerul de sistem.

Există *trei metode de optimizare* ce urmăresc micşorarea timpului consumat cu poziţionarea capetelor de citire/scriere *(seek)* la unităţile de discuri (componentă esenţială a duratei transferului de intrare/ieşire) :

— *căutarea celui mai apropiat cilindru (NEAR)*, metodă ce forţează alegerea operaţiei de intrare/ieşire plasată (ca adresă disc) cel mai aproape de cererea curentă (anterioară), indiferent de solicitarea acesteia ;

— *deplasarea înainte şi înapoi (ELEVATOR)*, metodă ce ia în considerare mai întîi cererile de intrare/ieşire într-o direcţie (de-a lungul discului) şi apoi în direcţia inversă ;

— *deplasarea pe cilindru (SCAN)*, metodă ce ia în considerare cererile de intrare/ieşire numai pe o direcţie, de la un cilindru inferior către altul superior.

Metodele de optimizare urmăresc luarea în considerare a cererii de intrare/ieşire cu „cea mai bună“ adresă disc. Cererile de prioritate mai mare sînt în continuare servite înaintea cererilor de prioritate mai mică ; performanţele de exploatare ale sistemului sînt îmbunătăţite prin reordonarea mai avantajoasă a cererilor de intrare/ieşire într-o gamă de priorităţi. Pentru limitarea numărului de treceri (desconsiderări) ale unei anumite cereri de intrare/ieşire, sistemul utilizează un contor de corectitudine a acceselor.

d) *Suprapunerea operaţiilor de poziţionare disc („seek“)*

Accesarea unui sector specific pe un volum disc are loc în mai multe etape :

— poziţionrea capului de citire/scriere (seek) pe cilindrul corespunzător ;

— aşteptare pînă la trecerea sectorului specificat prin faţa capului de citire/scriere (ca urmare a rotaţiei volumului disc) ;

— transferul efectiv al sectorului disc în/din memorie.

Primele două operaţii se pot suprapune între unităţile de disc ataşate aceluiaşi controlor. Această facilitate este utilizată de majoritatea driver-elor de disc în sistemul de operare **MIX-PLUS**, permiţînd suprapunerea operaţiilor de poziţionare disc („seek“) pe toate unităţile ataşate unui controlor de dispozitiv. În paralel cu activităţile de poziţionare poate avea loc un singur transfer de date la un moment dat. Acest mecanism este exploatat pe acele controloare de dispozitiv ce generează o întrerupere de atenţionare la terminarea unei operaţii de poziţionare.

Pentru obţinerea unui paralelism maxim, sistemul de operare **MIX-PLUS** implementează un mecanism de întîrziere a asignării controlorului dacă acesta era ocupat. Monitorul menţine o coadă de solicitări de acces la controlor şi plasează un bloc „fork“ în această coadă în cazul aşteptării driver-ului pe eliberarea controlorului. La eliberarea controlorului, Monitorul dă controlul primului driver în aşteptarea accesului.

e) *Dispozitive cu acces dual*

Dispozitivele cu *acces dual* conțin căi multiple de acces pentru funcțiile de control și transfer de informații. O unitate cu acces dual este conectată simultan la două controloare, avînd același număr de unitate fizică pentru fiecare controlor în parte.

Pentru a suporta operațiile cu acces dual, structurile de date asociate driverelor, specifice sistemului de operare **MIX-PLUS**, reflectă existența unor controloare alternate **(KRB)**. Pentru atribuirea unui controlor (asociată lui), driverul solicită Monitorului serializarea accesului la un controlor particular prin crearea unui proces „*fork*“ în coada asoaită blocului de acces la controlor **(KRB)**.

La căpătarea accesului din partea Monitorului, contextul driverului specific unei unități este asociat cu controlorul specificat. Driver-ul trebuie să capete acces la controlor înaintea accesării registrelor asociate în pagina externă. Întreruperile controlorului sînt direcționate către unitatea de dispozitiv implicată.

La terminarea operației de intrare/ieșire, driverul solicită din nou Monitorul pentru a elibera accesul la controlor.

11

Servicii de gestiune și manipulare a datelor

Gestiunea și manipularea datelor, pe dispozitivele de intrare/ieșire reprezintă întotdeauna o funcție importantă a unui sistem de operare. În cadrul familiei de sisteme de operare **MIX** s-au adoptat o serie de convenții și structuri care să asigure atît facilități simple de acces utilizator, cît și metode de acces sofisticate pentru diferite organizări logice ale datelor.

11.1. Unități de date fizice și logice

Cea mai mică unitate de transfer, între dispozitivele de intrare/ieșire (I/E) și memorie, este *înregistrarea fizică*. Lungimea înregistrărilor fizice este dependentă de tipul dispozitivului de I/E. Această lungime poate fi modificată de către utilizator cu ajutorul comenzilor operator **MCL/DCL (SET/BUF)**.

Blocul de date se poate defini ca fiind o înregistrare fizică în lungime de 512 octeți care se folosește la transferul între anumite dispozitive de I/E și memoria calculatorului. În general transferul pe blocuri se realizează pentru dispozitivele de I/E care folosesc suporturi magnetice.

O colecție de blocuri grupată pe un suport extern se numește *volum fizic*. Exemple de volume fizice sînt : pachetul de discuri, rola de bandă magnetică, caseta magnetică etc.

Unitățile de date logice sînt acele date folosite de către un utilizator, în programele sale, pentru a memora, transfera și primi informații. Aceste informații se deosebesc între ele prin anumite caracteristici logice, specifice fiecăreia (de ex. : tip de date, lungime etc. ...), iar aceste caracteristici nu depind de felul dispozitivului de I/E, ci ele sînt determinate de către utilizator.

Cîmpul este cea mai mică unitate de date logică. Lungimea cîmpului este stabilită de către utilizator în funcție de aplicația pentru care a fost scris programul său.

O colecți de cîmpuri, tratate ca o unitate, se numește *înregistrare logică*. Caracteristicile uni înregistrări logice sînt stabilite de către utilizator. însă ele pot fi definite și fizic. Astfel, o înregistrare logică poate ocupa unul sau mai multe blocuri sau pot exista mai multe înregistrări logice într-un bloc. Deseori în loc de înregistrare logică se mai folosește termenul de *„articol"* sau, pe scurt, *„înregistrare"* (*„record"*).

Mai multe înregistrări logice, grupate într-unul sau mai multe blocuri pe un suport, formează un *fișier* (*„file"*). Caracteristicile unui fișier sînt stabilite de către utilizator sau de către sistem.

O colecție de fișiere rezidentă pe un volum fizic se numește *volum logic*. Fișierele dintr-un volum logic pot avea, sau nu, caracteristici comune, altele decît acelea de a fi pe același suport.

11.2. Caracteristicile dispozitivelor de I/E

Pentru a citi/scrie fișiere cu ajutorul dispozitivelor de I/E se folosesc suporturi de fișier (de exemplu : cartela, hîrtia, banda de hîrtie, banda magnetidă, discul magnetic).

Indiferent de suportul fișierelor, în cadrul sistemului de operare **MIX**, dispozitivele de I/E sînt clasificate în :

1. dispozitive care suportă o structură de fișiere ;
2. Dispozitive care nu suportă o structură de fișiere.

Prin dispozitive de I/E care acceptă o structură de fișiere se înțeleg acele dispozitive pe care sistemul de operare **MIX** poate recunoaște un fișier în funcție de anumite caracteristici indicate de către utilizatorul sistemului.

Consola operator, imprimanta, lectorul de cartele sînt exemple de dispozitive de I/E cu suporturi nereutilizabile care nu permit identificarea fișierelor în funcție de anumite caracteristici. De asemenea, aceste dispozitive pot să transfere datele numai în secvența în care ele apar fizic. Deci singurul mod de acces posibil la articolele scrise/citite pe/de pe aceste dispozitive este *accesul secvențial*.

Spre deosebire de acestea, discul magnetic și banda magnetică permit identificarea fișierelor, în funcție de anumite caracteristici, de pe un volum logic. Aceste dispozitive de I/E permit citirea articolelor dintr-un fișier atît în *acces secvențial* cît și în *acces direct* (numai pentru disc). Totodată, pe discul magnetic se pot face și scrieri de articole în *acces direct* sau secvențial, iar pe celelalte dispozitive scrierea se poate face numai secvențial.

11.3. Organizarea logică a datelor

O caracteristică importantă a familiei de sisteme de operare **MIX** o constituie reflectarea caracteristicilor fizice ale unui dispozitiv de I/E într-o organizare logică a datelor. Această acțiune este îndeplinită de Sistemul de Gestiune a Fișierelor (**SGF**) **MIX** ce asigură interfețe specifice de acces către programul utilizator.

Utilizarea facilităţilor de acces oferite de **SGF** permite tratarea uniformă a tuturor dispozitivelor cu structură de fişiere. Aceste dispozitive apar programelor utilizator ca fiind organizate în fişiere, formate din blocuri consecutive de 512 octeţi fiecare, numerotate de la zero pînă la ultimul bloc din fişier. În realitate însă, blocurile pot fi distribuite pe suportul de dispozitiv, iar uneori, dimensiunea unei înregistrări fizice pe dispozitiv, poate să fie diferită de 512 octeţi.

În terminologia **MIX**, înregistrările fizice de pe un dispozitiv de exemplu, sectoarele unui disc) se numesc *blocuri fizice*. La nivelul interfeţei cu dispozitivul hardware, driver-ul de I/E mapează blocurile fizice în *blocuri logice*. Blocurile logice sînt numerotate ca şi blocurile fizice, începînd cu blocul zero (primul bloc de pe dispozitiv) şi terminînd cu ultimul bloc de pe dispozitiv.

La nivelul interfeţei utilizator, Sistemul de gestiune a fişierelor mapează blocurile logice în *blocuri virtuale*. Blocurile virtuale, numerotate de la 1 pînă la sfîrşitul unui fişier, sînt relative la un fişier, pe cînd blocurile logice sînt relative la un volum.

11.4. Structuri de fişiere şi metode de acces

Structura de fişiere reprezintă modalitatea de organizare a înregistrărilor logice în fişiere ; ea descrie localizarea fizică relativă a blocurilor ce compun un fişier.

Metoda de acces reprezintă un set de reguli utilizate la selectarea înregistrărilor logice într-un fişier. Cea mai simplă metodă de acces este cea secvenţială : fiecare înregistrare este prelucrată în ordinea în care apare în fişier. O altă metodă cunoscută este cea a accesului direct : fiecare înregistrare trebuie precizată pentru a fi accesată.

Un dispozitiv cum este banda magnetică poate fi prelucrat numai secvenţial. Un dispozitiv cum este discul magnetic poate fi prelucrat în ambele moduri, accesul direct avînd însă un avantaj net superior accesului secvenţial.

Structurile de fişiere şi metodele de acces utilizate pe larg în sistemul de operare **MIX** sînt următoarele :

Structură	*Metodă de acces*
Înlănţuită	Secvenţial
Contiguă	Secvenţial sau acces direct
Mapată (ierarhică)	Secvenţial sau acces direct

Structura înlănţuită este formată dintr-o serie de blocuri fizic neadiacente pe acelaşi dispozitiv. Pentru conectare, fiecare bloc conţine un *pointer* către următorul (ultimul fiind zero). Această structură este utilă pentru acele fişiere la care nu se cunoaşte dimensiunea finală. Ea permite extensii ulterioare şi utilizează eficient spaţiul pe dispozitivele de tip disc.

Structura contiguă este formată dintr-o serie de blocuri fizic adiacente pe dispozitivul de înregistrare. Structura este utilă în cazul prelucrărilor cu acces direct, întrucît ordinea blocurilor nu impune ordinea de prelucrare a datelor. Determinarea localizării fizice a unui bloc se face fără a referi alte blocuri în fișier.

Structura mapată (ierarhică) constă dintr-un set de fișiere virtual contigue ; ea apare programului utilizator ca un set de blocuri adiacente, direct adresabile. Întrucît majoritatea fișierelor nu pot ocupa spații contigue pe dispozitiv și blocurile aparținînd fișierelor sînt alocate în mod dispersat, se folosesc structuri suplimentare, numite *blocuri de descriere (antete de fișiere — „file header")*, ce descriu identitatea blocurilor ce constituie fișierele. Această metodă utilizează eficient spațiul disponibil pe un dispozitiv de tip disc și permite extensia ușoară a fișierelor, impunînd o interfață de programare uniformă. O structură mapată poate fi creată și sub forma unui fișier contiguu pentru a permite un acces direct rapid.

Structurile de fișiere prezentate mai sus pot fi modificate sau combinate pentru a obține o nouă metodă logică de prelucrare.

Structura de fișiere cea mai generală și flexibilă este *structura indexată* formată din două fișiere contigue. Primul fișier constituie o mapă ordonată a celui de-al doilea fișier ce conține date. Mapa sau porțiunea de index conține fie o listă ordonată de chei selectate din înregistrările de date, fie pointeri către înregistrările de date. fie combinația acestora.

Înregistrările fișierului de date pot fi prelucrate în ordinea porțiunii de index sau înregistrările de date pot fi selectate căutîndu-se în porțiunea de index cheia de identificare a acestor înregistrări. Aceste metode de prelucrare logică se numesc *acces secvențial indexat* sau, respectiv, *acces direct după cheie.*

11.5. Fișiere catalog și metode de acces a acestora

Așa cum structura de fișiere și metodele de acces sînt necesare pentru localizarea înregistrărilor în fișiere, structurile de cataloage și metodele de acces ale acestora sînt necesare pentru localizarea fișierelor pe volume.

Un *catalog („directory")* reprezintă o structură sistem utilizată pentru organizarea unui volum în fișiere. Ea permite unui utilizator localizarea unui fișier fără a preciza adresa fizică a acestuia, reprezentînd o metodă de acces direct aplicată volumelor pentru localizarea fișierelor.

Sistemul de operare **MIX** utilizează două tipuri diferite de cataloage pentru diferențierea fișierelor între utilizatori : un catalog principal (**MFD** — *Master File Directory)* și un număr variabil de cataloage utilizator (**UFD** — *User File Directories).* Acestea sînt la rîndul lor fișiere înregistrate pe volumul pentru care îndeplinesc funcțiile de cataloage.

La crearea unui fișier, sistemul înscrie numele acestuia într-un catalog de fișiere utilizator (**UFD**) și memorează codul de identificare al utilizatorului (**UIC**) în antetul de fișier pentru a indica proprietarul acestuia. În cele mai multe cazuri, **UFD**-ul corespunde cu **UIC**-ul proprietarului ; este

posibil, însă, ca un fișier să se afle într-un catalog ce nu are legătură cu codul proprietarului. Este de asemenea, posibil ca un fișier să apară simultan în mai multe cataloage.

Catalogul de fișiere utilizator este el însuși un fișier, care trebuie creat în mod explicit prin intermediul comenzii operator **UFD.** Catalogul se specifică sub forma unui **UIC**, deci [*g*, *m*], unde :

g — codul grupului de utilizatori (număr octal din intervalul 1—377);
m — codul membrului din cadrul grupului (proprietarul — număr octal cuprins între 1 și 377).

Numele efectiv al fișierului catalog reprezintă o concatenare a codurilor de grup și de proprietar, terminate cu tipul **DIR.** De exemplu, numele fișierului de directori care corespunde **UIC**-ului **[115, 23]** *este* **115023.DIR.** De remarcat că se adaugă zerouri pentru ca fiecare (*g* și *m*) să conțină cîte trei cifre.

Toate fișierele catalog **(UFD)** sînt la rîndul lor inventariate în fișierul catalog principal **(MFD).** El este unic per volum și corespunde **UIC**-ului [0, 0], numindu-se de aceea 000000.**DIR.** Fișierele de catalog (atît cele pentru fișiere utilizator, cît și cel principal) cuprind nume de fișiere și adresele antetelor acestor fișiere. Antetul unui fișier conține informații referitoare la proprietarul fișierului și la plasarea fizică pe volum a segmentelor fișierului ; antetul ocupă unul sau mai multe blocuri în fișierul de indecși al volumului ([0, 0] **INDEXF.SYS**).

Pentru localizarea unui fișier, sistemul verifică în catalogul principal **(MFD)** localizarea catalogului utilizator asociat **(UFD)** și apoi localizarea fișierului în cadrul acestuia.

11.6. Protecția fișierelor

Sistemul de operare **MIX** asigură o schemă de protecție unică la nivelul volumelor și fișierelor. Atributele de protecție pot fi specificate pentru întregul volum ca și pentru fișierele aparținînd volumului.

Pentru a putea accesa un fișier sînt necesare următoarele acțiuni :
— volumul pe care se află acesta trebuie să fie montat logic prin comanda **MOU** ;
— trebuie specificat catalogul, în care se află fișierul ;
— trebuie satisfăcute condițiile din masca de protecție asociată fișierului.

Fiecare fișier are o mască de protecție care descrie tipul de acces permis pentru fiecare din cele patru grupe de utilizatori. Cele *patru tipuri de acces* sînt :

— **R** — citire
— **W** — scriere
— **E** — extindere
— **D** — ștergere

Cele *patru grupuri de utilizatori* sînt definite după criteriul codului lor de identificare **(UIC)** :

— *sistem :* — toate task-urile privilegiate, adică acelea care se execută sub un **UIC** avînd codul de grup mai mic sau egal cu 10 (octal)

— *proprietar* : — task-uri care se execută sub un **UIC** identic cu cel al proprietarului
— *grup* : — orice task care se execută sub un **UIC** care are același cod de grup cu proprietarul fișierului
— *restul populației* : — orice alt task sau utilizator care nu face parte din nici una din categoriile de mai sus.

Drepturile de acces la un volum, respectiv fișier, se stabilesc în unul din următoarele moduri :

— la inițializarea volumului prin comanda operator **INI** se poate specifica masca de protecție implicită pentru toate fișierele ce vor fi create ulterior pe volum ;

— la crearea unui fișier catalog (**UFD**), prin comandă operator se pot stabili drepturile de acces asociate catalogului. Un utilizator permite de obicei acces în citire la un fișier catalog, refuzînd însă accesul în scriere la fișierele conținute în el ;

— prin intermediul programului utilitar **PIP**, proprietarul fișierului poate modifica drepturile de acces la fișier ;

— la montarea logică a unui volum, prin comanda operator **MOU** se poate specifica protecția implicită pentru fișierele ce vor fi create pe volum. Această protecție prevalează asupra protecției specificate la inițializarea volumului.

Pentru a putea accesa un fișier, task-ul solicitant trebuie să satisfacă masca de protecție atît pentru fișier, cît și pentru catalogul în care este conținut fișierul. De exemplu, pentru a putea scrie într-un fișier deja existent, este necesar cel puțin acces în citire la fișierul catalog și acces în scriere la fișier. Pentru a putea crea un fișier trebuie avut acces la fișierul catalog (**UFD**) atît în scriere, cît și în extensie.

11.7. Specificatorul de fișier

În familia de sisteme de operare **MIX**, fișierele organizate pe diferite suporturi sînt identificate în funcție de caracteristicile lor, caracteristici specificate de către utilizator la crearea fișierului. Aceste caracteristici sînt indicate Sistemului de Gestiune a Fișierelor (**FCS**) prin intermediul unor structuri de date specifice, de exemplu : blocul de identificare implicită a fișierului, descriptorul de fișier etc.

Totalitatea caracteristicilor unui fișier, furnizate modulelor de acces **FCS**, într-un anumit format și într-o anumită ordine, poartă numele de *specificator de fișier*. În general, formatul unui specificator de fișier este următorul :

Nume-dispozitiv : [*Catalog*] *Nume-fișier. Tip-fișier ; Versiune-fișier* în care :
— *Nume dispozitiv :* reprezintă numele dispozitivului (fizic sau logic) și unitatea pe care se află volumul conținînd fișierul. Numele unui dispozitiv de I/E se compune din :

— un șir de caractere ASCII, cu lungimea de 2 octeți, prin care se specifică numele perifericului (de exemplu **MT, DK, CR, LP** etc.) ;

— una pînă la trei cifre (octale) prin care se specifică numărul unității pe/de pe care se scrie/citește un fișier. Dacă nu se specifică acest număr se consideră, implicit, unitatea cu numărul 0.

Numele dispozitivului de I/E, indiferent dacă numărul unității este prezent sau nu, se termină totdeauna cu caracterul „:". În cazul în care numele dispozitivului lipsește, se consideră dispozitivul implicit **SY:**

[*Catalog*] sau ⟨*catalog*⟩ — reprezintă catalogul de fișiere utilizator **(UFD)** în care este plasat fișierul specificat (pentru volume disc cu structură **FILES11)** ; acest cîmp nu are semnificație pentru volumele de bandă magnetică cu structură **ANSI.**

Pentru *cataloagele în format numeric,* specificatorul de catalog este de forma *codului de identificare utilizator* **(UIC)** :

[*g*, *m*] sau ⟨*g*, *m*⟩, în care :

- **g** și **m** reprezintă numere octale de la 0 la 377 reprezentînd respectiv, codul de grup și codul de membru al proprietarului de fișier ;
- parantezele drepte ([ ]), unghiulare (⟨ ⟩) și virgula (,) fac parte din sintaxa de apel.

În locul parametrilor **g** sau **m** se poate folosi specificatorul implicit „*wildcard*" (*), următoarele substituții fiind corecte :

[*g*, *] — indică toți membrii din grupul *g* ;
[*, *m*] — indică toate grupurile cu membrii *m* ;
[*] — indică toate cataloagele ;
[ ] — indică catalogul implicit curent.

Cataloagele în format numeric sînt disponibile sub sistemele de operare **MIX/MIX-PLUS/MIX-RT.**

În sistemul de operare **MIX-PLUS** se pot utiliza și *cataloage cu nume,* de forma :

[*numecatalog*] sau ⟨*numecatalog*⟩, în care :

- *numecatalog* reprezintă un șir de caractere alfanumerice pînă la nouă caractere ;
- parantezele drepte ([ ]) sau unghiulare (⟨ ⟩) fac parte din sintaxa de apel ;
- crearea cataloagelor cu nume este posibilă cu comanda **MCL, UFD** sau **DCL CREATE/DIRECTORY,** după validarea acestora (prin comanda **MCL SET/NAMED** sau **DCL SET DEVICE/NAMED-DIRECTORY)** ; invalidarea creării de cataloage cu nume este posibilă prin comanda **MCL SET/NONAMED** sau **DCL SET DEVICE-NONAMED-DIRECTORY** ;
- cataloagele în format numeric pot fi echivalente și cu cataloage cu nume, de exemplu :
  [5,73] sau [005073]

Stabilirea catalogului implicit (numeric sau cu nume), la nivelul interfeței operator, este posibilă prin comanda **MCL SET/DEF** sau **DCL SET/DEFAULT.**

— *Nume-fișier :* specifică numele unui fișier. Este format dintr-un șir de la 0 pînă la 9 caractere alfanumerice, plus caracterele speciale de concatenare (—) și dolar ($). Se admit și caractere mici care sînt convertite automat, în mari, de modulele de acces **FCS.**

În locul cîmpului nume-fișier se poate folosi și specificatorul implicit „*wildcard*“ (*), următoarele substituții fiind corecte :

* . ; — indică toate fișierele ;

. ; — indică un nume nul, cu lungimea zero (*0*).

Numele de fișier este separat de tip printr-un *punct.*

— *Tip :* constă dintr-un punct (.), urmat de o mnemonică de 0 la 3 litere care identifică conținutul fișierului.

Se admit și caractere mici care sînt convertite automat în mari de modulele de acces **FCS.** În locul cîmpului tip se poate folosi și specificatorul implicit „*wildcard*“ (*), următoarele substituții fiind corecte :

.* — indică toate tipurile

. — indică *tip nul.*

Tipul fișierului este separat de numărul de versiune prin caracterul *punct și virgulă* (;). Tipul implicit depinde de task-ul căruia îi este destinată linia de comandă și de faptul că specificatorul se referă la un fișier de intrare sau de ieșire. De exemplu, macroasamblorul **MAC** consideră că tipul implicit pentru fișierul de intrare este **.MAC,** iar pentru cel de ieșire **.OBJ.**

— *Versiune :* reprezintă un număr octal de la 0 la 77777 care diferențiază între ele diverse versiuni ale unui fișier. De exemplu, la crearea unui fișier sistemul îi dă numărul de versiune *1.*

Cînd fișierul este deschis pentru editare (cu programul utilitar **EDT),** sistemul creează un nou fișier. Acesta are același nume și același tip, dar numărul de versiune este mai mare cu unu.

Numerele —1 și 0 au o semnificație specială : —*1* specifică cea mai mică versiune existentă a unui fișer, iar *0* (versiune nulă) cea mai mare versiune existentă.

Pentru fișierele de intrare se consideră implicit cea mai mare versiune existentă ; pentru fișierele de ieșire — cea mai mare versiune existentă plus 1.

În cazul utilizării parametrului implicit, „*wildcard*“ (*), se iau în considerație toate versiunile de fișier.

În sistemul de operare **MIX-PLUS,** numărul de versiune poate fi precizat și în zecimal, în gama 0 la 32767.

Sistemele de operare **MIX/MIX-PLUS** pun la dispoziția utilizatorilor un set de *tipuri standard de fișiere* care să reflecte conținutul efectiv al fișierelor. Ele sînt utilizate de toate componentele sistem. Deși orice combinație de 3 litere poate deveni tip de fișier, utilizarea tipurilor standard ușurează lucrul cu sistemul de operare. Principalele tipuri utilizate sînt prezentate mai jos.

Tipul	*Conținutul fișierului*
BAS	Program sursă în limbaj **BASIC**
BAT	Fișier cu comenzi batch
B2S	Program sursă în limbaj **BASIC-PLUS-2**
CBL	Program sursă în limbaj **COBOL**
CLB	Bibliotecă cu fișiere de comenzi indirecte
CMD	Fișier de comenzi indirecte
COR	Fișier de corecții **SLP**
DAT	Fișier de date (prin opoziție cu un program)

DIR	Fişier de cataloage (de exemplu, un **UFD**)
FTN	Program sursă în limbaj **FORTRAN**
LOG	Fişier de înregistrare a unei sesiuni
LST	Listing
MAC	Program sursă în limbaj de asamblare
MAP	Hartă de alocare a memoriei produsă de **TKB**
MLB	Bibliotecă de macroinstrucţiuni
OBJ	Modul obiect (rezultat al unei asamblări sau compilări)
ODL	Descriptor de structuri segmentate pentru **TKB**
OLB	Bibliotecă de module obiect
PAT	Fişier sursă cu corecţii utilizate de **PATCH**
POB	Modul obiect cu corecţii aplicate de **PATCH**
SML	Bibliotecă sistem de macroinstrucţiuni
STB	Tabelă de simboluri
SYS	Imagine sistem încărcabilă
TMP	Fişier temporar
TSK	Imagine task
TXT	Fişier text
ULB	Bibliotecă universală.

Pentru fişierele pe bandă magnetică există şi un format alternativ al specificatorului de fişier :

Nume-dispozitiv : [*catalog*] *Şir-între-ghilimele ; Versiune-fişier.*

Nume-dispozitiv, *catalog* şi *Versiune-fişier* au semnificaţiile prezentate anterior.

Şir-între-ghilimele reprezintă un şir pînă la 17 caractere alfanumerice, cuprins între *ghilimele* (“), ce se substituie elementelor *Nume-fişier* şi *Tip-fişier* din specificatorul de fişier. Ex. :

MM3 : “Fişier, de — TEST“ specifică :

Toate versiunile fişierului (Fişier de TEST), pe dispozitivul MM3 :

Un specificator de fişier (disc sau bandă) poate substitui, parţial sau total, elementele sale cu *nume logice.*

11.8. Utilizarea numelor logice în sistemul de operare MIX-PLUS

În sistemul de operare **MIX-PLUS** există posibilitatea utilizării unor nume logice drept substitut al elementelor sau întregului specificator de fişier. Independenţa programelor de suporturile de I/E şi de specificatorii de fişiere este asigurată, în mod complet, prin utilizarea numelor logice.

La scrierea unui program în limbajul **MACRO**, referirea unui fişier de I/E se face utilizînd un nume logic sugestiv (de ex., **INFILE** pentru un fişier de intrare, **OUTFILE** pentru un fişier de ieşire, **SYS$LOGIN** pentru identificarea dispozitivului sistem utilizator, **SYS$LIBRARY** pentru identificarea

bibliotecilor sistem etc.). După asamblare și link-editare, dar înainte de lansarea în execuție, se utilizează comenzi operator **MCL** și **DCL (DEFINE** și **ASSIGN)**, prin care se asociază numele logice utilizate în program cu fișierele sau dispozitivele precise dorite de către programul de aplicație. Aceste asocieri se pot schimba, succesiv, la momente de timp diferite ca urmare a schimbării suporturilor de I/E sau indisponsibilizării acestora.

Evidența perechilor nume logice — nume echivalente se păstrează în 3 tabele de nume logice :

— tabela numelor logice asociate task-urilor ;
— tabela numelor logice asociate grupurilor de task-uri ;
— tabela numelor logice sistem.

Numele logice și șirurile lor echivalente sînt formate din 1 la 63 caractere alfanumerice, inclusiv caracterele speciale *dolar* ($) și *concatenare* (-). Șirurile echivalente trebuie să conțină punctuația specifică identificării tuturor elementelor unui fișier (paranteze drepte și unghiulare, virgulă, punct, punct și virgulă, două puncte). Dacă șirul echivalent reprezintă un nume de dispozitiv, acesta trebuie terminat obligatoriu de caracterul (:). Dacă specificatorul de fișier folosește numai parțial nume logice, acestea trebuie să identifice numai elementele din stînga ale specificatorului.

Analiza și translatarea unui specificator de fișier ce folosește (parțial sau total) nume logice este posibilă prin utilizarea directivelor Monitor :

— **PFCS$** — parsare specificator de fișier și construire bloc(uri) de descriere **FCS (FDB, DFNB** etc.) ;
— **PRMS$** — parsare specificator de fișier și construire bloc(uri) de descriere **RMS (FAB, RAB** etc.) ;
— **ACHN$** — permite asignarea unui număr logic unui dispozitiv de I/E descris cu nume logic.

Directivele **PFCS$** și **PRMS$** efectuează funcții de analiză și translatare complexe, fiind utilizate de rutinele de acces **FCS** și **RMS.** Procesul de translatare a numelor logice poate fi iterativ, continuînd pînă la identificarea completă a tuturor elementelor specificatorului de fișier. Translatarea se oprește pe eroare, la depășirea a zece nivele de substituire sau la detectarea unor asignări circulare.

În timpul procesului de translatare au loc următoarele operații :

— parsarea specificatorului de fișier (utilizînd directivele **PFCS$** și **PRMS$**) ;
— expandarea șirurilor translatate ;
— comasarea șirurilor translatate cu șirurile specificate implicit (pentru elementele lipsă din specificatorul de fișier). Identificarea elementelor lipsă se face utilizînd directiva Monitor **FSS$** (scanare specificator fișier).

12

Sistemul de gestiune a fișierelor (FCS)

12.1. Prezentare generală

Sistemul de gestiune a fișierelor (**FCS** — File Control Services), suportat de familia de sisteme de operare **MIX**, permite utilizatorilor să lucreze cu un volum mare de date stocate pe diverse suporturi.

Pentru aceasta Sistemul de Gestiune a Fișierelor asigură următoarele funcții principale :

— acces la nivel de fișier și articol ;

— acces secvențial sau direct la informațiile de pe suport ;

— controlul logic al fișierelor pe dispozitivele de intrare/ieșire : deschidere/închidere/ștergere fișier, sincronizare operații de transfer ;

— gestiunea zonelor tampon ;

— blocarea/deblocarea articolelor în zonele tampon **FCS** ;

— protecția fișierelor pe suport ;

— asigură independența programelor față de suport.

Dacă primele 6 funcții sînt comune oricărui sistem de gestiune a fișierelor, cea de-a șaptea, independența față de suport, este o funcție care oferă sistemului de gestiune a fișierelor **MIX (FCS)**, o importanță deosebită pentru componentele sistemului. Aceasta deoarece toate programele utilitare precum și alte componente ale sistemului de operare, folosesc Sistemul de Gestiune a Fișierelor, prin intermediul unor rutine de acces, în efectuarea operațiilor de I/E pentru a asigura o tratare unitară a informațiilor de pe suporturi de dispozitiv.

Independența programelor față de suport se poate defini ca acea proprietate a unui program de a efectua operații de I/E cu orice dispozitiv de I/E fără a se schimba nimic în codul programului. Singura componentă a sistemului de operare **MIX** care poate oferi această proprietate unui program utilizator este Sistemul de Gestiune a Fișierelor **FCS.**

Rutinele de acces **FCS** permit citirea și scrierea în fișiere aflate pe volume cu structura simplă sau complexă de fișiere. Ele prelucrează fișierele

ca pe o succesiune (colecţie) de înregistrări logice. Datele se pot scrie într-un fişier, într-un mod care să permită regăsirea ulterioară a lor, fără ca utilizatorul să fie obligat să cunoască formatul în care acestea au fost scrise. Utilizarea rutinelor **FCS** asigură un grad înalt de transparenţă pentru programele utilizator, înregistrările de date putînd fi citite sau scrise ca unităţi logice dependente de cerinţele unei aplicaţii particulare.

Sistemul de gestiune a fişierelor **FCS** „vede" conţinutul unui fişier ca o secvenţă continuă de înregistrări ale utilizatorului.

Dimensiunea fiecărei înregistrări este determinată de utilizator şi nu de mediul fizic de memorare.

Rutinele **FCS** permit două moduri de acces pentru scrierea şi regăsirea înregistrărilor :

— în *modul secvenţial*, o înregistrare este scrisă (în mod fizic) imediat alături de cele precedente ; înregistrările sînt regăsite pe baza acestei adiacenţe ;

— în *modul aleator*, un task poate citi şi scrie înregistrări specificînd poziţia lor relativă în fişier. Acest mod este permis numai pentru fişierele cu înregistrări de lungimi egale, aflate pe disc.

Pentru apelarea funcţiilor **FCS**, programul utilizator foloseşte apeluri de macroinstrucţiuni ce specifică operaţiile dorite asupra structurii de fişiere. Interfaţa programelor utilizator cu sistemul de intrare/ieşire este simplificată de existenţa acestui set extensiv de macroinstrucţiuni, ce poate fi împărţit în următoarele 4 categorii :

• *Macroinstrucţiuni de iniţializare*, ce sînt utilizate pentru stabilirea structurilor de descriere a datelor şi a zonelor tampon folosite pentru operaţiile de I/E ;

• *Macroinstrucţiuni de prelucrare*, ce determină modul în care se face exploatarea fişierelor : acces secvenţial sau direct, operaţii la nivel de bloc sau înregistrare ;

• *Macroinstrucţiuni de prelucrare a liniilor de comandă*, ce asigură prelucrarea dinamică a liniilor de comandă introduse de la terminal. Macroinstrucţiunile apelează două rutine specifice din biblioteca sistem cu specificatorul **LB : [1, 1] SYSLIB.OLB**, capabile să prelucreze şiruri de linii de comandă :

— **GCML** — ce realizează toate funcţiile logice asociate introducerii de şiruri de comandă de la un terminal sau dintr-un fişier de comenzi ;

— **CSI** — ce preia şirurile de comandă din buferul de intrare **GCML** şi îl interpretează (parsează) pentru completarea unui bloc descriptor de fişier **(DSD)**, necesar modulelor de acces **FCS** la deschiderea (accesarea) fişierului specificat.

• *Macroinstrucţiunea* **CALL**, folosită pentru apelarea unor rutine specifice **FCS** : căutare fişier, inserare, ştergere intrare în catalog, schimbare nume fişier, extensie fişier etc.

12.2. Organizarea fișierelor FCS

Un fișier reprezintă o colecție de articole sau blocuri tratate unitar. Modul în care modulele de acces **FCS** construiesc un fișier pe suport se numește organizare de fișier. Din acest punct de vedere Sistemul de Gestiune a Fișierelor (**FCS**) din sistemul de operare **MIX** acceptă două tipuri de organizări, după cum urmează :

Organizarea secvențială a fișierelor

Un *fișier de organizare secvențială* este acel fișier în care articolele apar într-o secvență continuă. Ordinea articolelor în această secvență este ordinea în care articolele au fost scrise inițial de către utilizator.

Fișierele de organizare secvențială vor fi dotate sau nu cu identificatori de fișier în funcție de tipul suportului (dacă acceptă sau nu structură de fișier). Accesul la înregistrare este secvențial, unitatea de acces fiind înregistrarea. Lungimea înregistrării este stabilită de către utilizator sau, prin lipsă, de către sistem. Formatul înregistrărilor poate fi fix sau variabil. Înregistrările pot fi blocate sau nu, în funcție de lungimea definită de către utilizator.

În Fig. 12.1 este ilustrată organizarea secvențială a fișierelor pe suport magnetic și pe suport nemagnetic.

Organizarea nedefinită a fișierelor

O colecție de blocuri fizice de date, tratate unitar, definește un *fișier cu organizare nedefinită*. Fișierele de organizare nedefinită vor fi dotate sau nu cu identificatori de fișiere, în funcție de tipul suportului (dacă acceptă sau nu structură de fișier). Unitatea de acces este blocul. În cazul suporturilor ce nu permit înlocuirea de bloc, modulele de acces **FCS** permit accesul la blocul următor sau, dacă fișierul are identificator, permit accesul la un bloc de adresă dată, adresă relativă la începutul fișierului. În cazul suporturilor ce permit înlocuirea de bloc, modulele de acces **FCS** permit accesul, atît

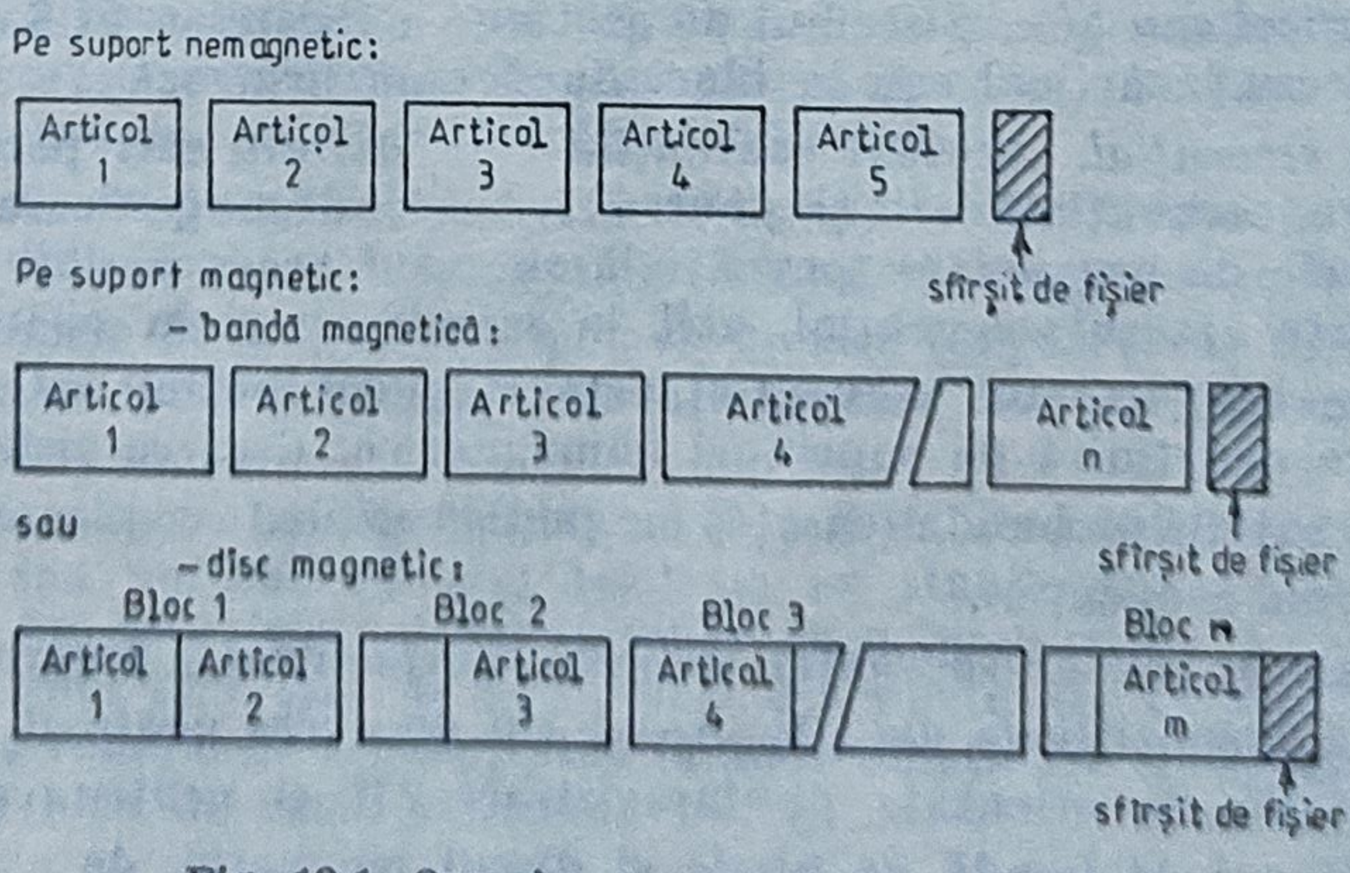


Fig. 12.1. Organizarea secvențială a fișierelor

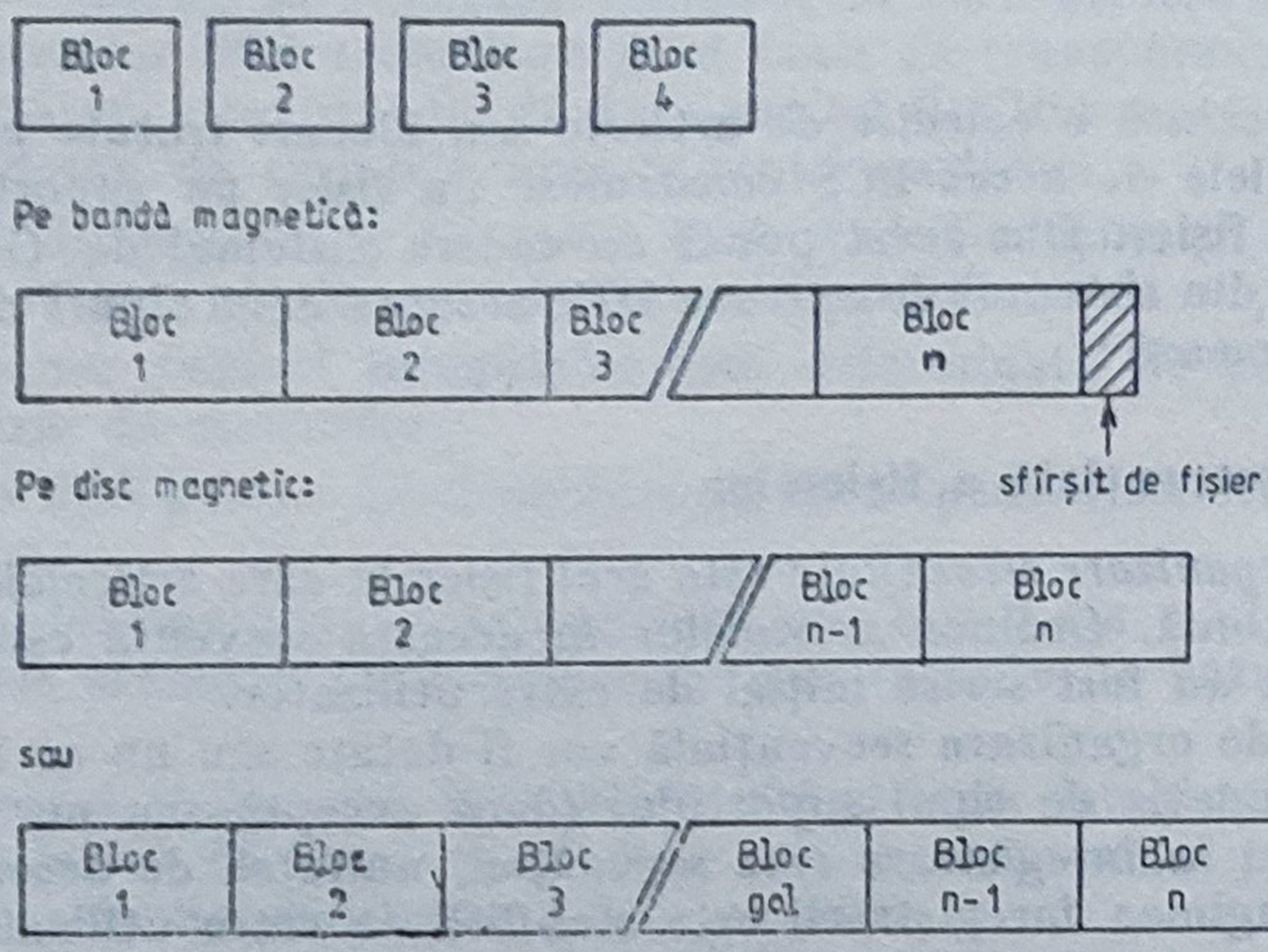


Fig. 12.2. Organizarea nedefinită a fișierelor

în scriere cît și în citire, la un bloc de adresă dată, adresă relativă la începutul fișierului (bloc virtual). Fig. 12.2 ilustrează organizarea nedefinită a fișierelor pe diverse suporturi. Acest tip de organizare a fișierelor nu presupune ca modulele de acces **FCS** să efectueze operații de blocare/deblocare. Pentru suporturile care nu permit accesul aleatoriu la informații, ordinea blocurilor scrise pe suport respectă întrutotul ordinea în care informațiile au fost furnizate de către utilizator.

12.3. Moduri de acces la fișiere FCS

Organizarea unui fișier presupune și definirea tehnicii de regăsire sau de scriere a datelor în fișier. Această tehnică este cunoscută sub numele de *mod de acces la articol sau bloc*. Sistemul de gestiune a fișierelor **FCS** acceptă trei moduri de acces la articol sau la bloc, după cum urmează :

• *Acces secvențial*. Accesul secvențial este definit atît pentru fișierele de organizare secvențială cît și pentru fișierele de organizare nedefinită. Pentru fișierele de organizare secvențială singurul acces posibil la articolele unui fișier este accesul secvențial, atît în scriere cît și în citire.

De asemenea, accesul secvențial este singurul posibil pentru fișierele de organizare nedefinită pe suporturi nemagnetice. Crearea fișierelor de organizare secvențială pe banda (casetă) magnetică se poate realiza de asemenea, numai în acces secvențial.

Utilizarea accesului secvențial asigură independența programelor utilizator față de dispozitivele de I/E. Acest tip de acces poate fi utilizat atît pentru dispozitivele orientate pe înregistrare cît și pentru cele orientate pe bloc (cititorul de bandă de hîrtie și discul magnetic, de exemplu).

• *Acces direct sau aleator (random).* Pentru suporturile magnetice se poate face citirea articolelor, blocurilor, de pe suport și în acces direct. Accesul direct este permis și în scrierea fișierelor dar numai pe disc magnetic. În principal, accesul direct a fost definit numai pentru fișierele de organizare nedefinită. Acest acces se realizează prin indicarea, de către utilizator, a adresei relative a blocului dorit față de începutul fișierului. Primul bloc din fișier se consideră că are adresa relativă 1.

Utilizarea accesului direct este permisă numai pentru dispozitivele orientate pe bloc.

• *Acces dinamic.* În funcție de aplicație, se poate imagina accesul dinamic la informațiile dintr-un fișier. Astfel, folosind posibilitățile oferite de către modulele de acces **FCS** de a modifica dinamic informațiile din **FDB**, se poate începe citirea unui fișier în acces secvențial cu macroinstrucțiunea **GET$** și, dacă nu ne interesează modul în care au fost blocate articolele, se poate indica citirea unui bloc, altul decît cel următor, folosind macroinstrucțiunea **READ$**.

12.4. Formatul datelor pe volume cu structură de fișier

Datele sînt transferate între dispozitivele periferice și memorie pe *blocuri*. Un *fișier* constă din *blocuri virtuale*, fiecare dintre ele putînd conține unul sau mai multe *articole logice* create de programul utilizator.

În accepțiunea *Sistemului de Gestiune a Fișierelor*, un bloc virtual dintr-un fișier are o lungime de 512 octeți și este numerotat secvențial începînd cu 1. Un fișier conține cel puțin un bloc virtual, iar un bloc virtual poate să conțină unul sau mai multe articole. Lungimea articolelor din blocurile virtuale se găsește sub controlul programului utilizator.

Articolele pot avea următoarele formate (fig. 12.3):

— *fix*, în care toate articolele din fișier sînt de lungime egală;

— *variabil*, în care articolele unui fișier nu au aceeași lungime, primii 2 octeți ai articolului conținînd în acest caz lungimea, exprimată în număr de octeți (lungime ce nu va cuprinde primii 2 octeți);

— *variabil secvențat*, în care între cei 2 octeți, ce conțin lungimea articolului, și corpul articolului, se mai găsesc 2 octeți în care se înscrie un număr de secvență precizat de către utilizator.

Articolele de lungime fixă sau variabilă sînt aliniate la multiplu de cuvînt. Octetul final ca urmare a unei lungimi impare este ignorat.

Pe volumele disc, articolele pot să încalce (depășească) limitele de bloc (începutul unui articol este plasat într-un bloc iar sfîrșitul articolului în blocul următor).

Pe volumele de bandă magnetică pot fi de asemeni definite articole cu format fix sau variabil (format conform cu standardul ANSI). Pe bandă magnetică, un bloc virtual corespunde cu o înregistrare fizică. Lungimea implicită a unui bloc este de 512 octeți. Ea poate lua valori de la 8 octeți (14 pentru o funcție de scriere) pînă la 2048 octeți. Nu se admite încălcarea limitelor de bloc de către articole.

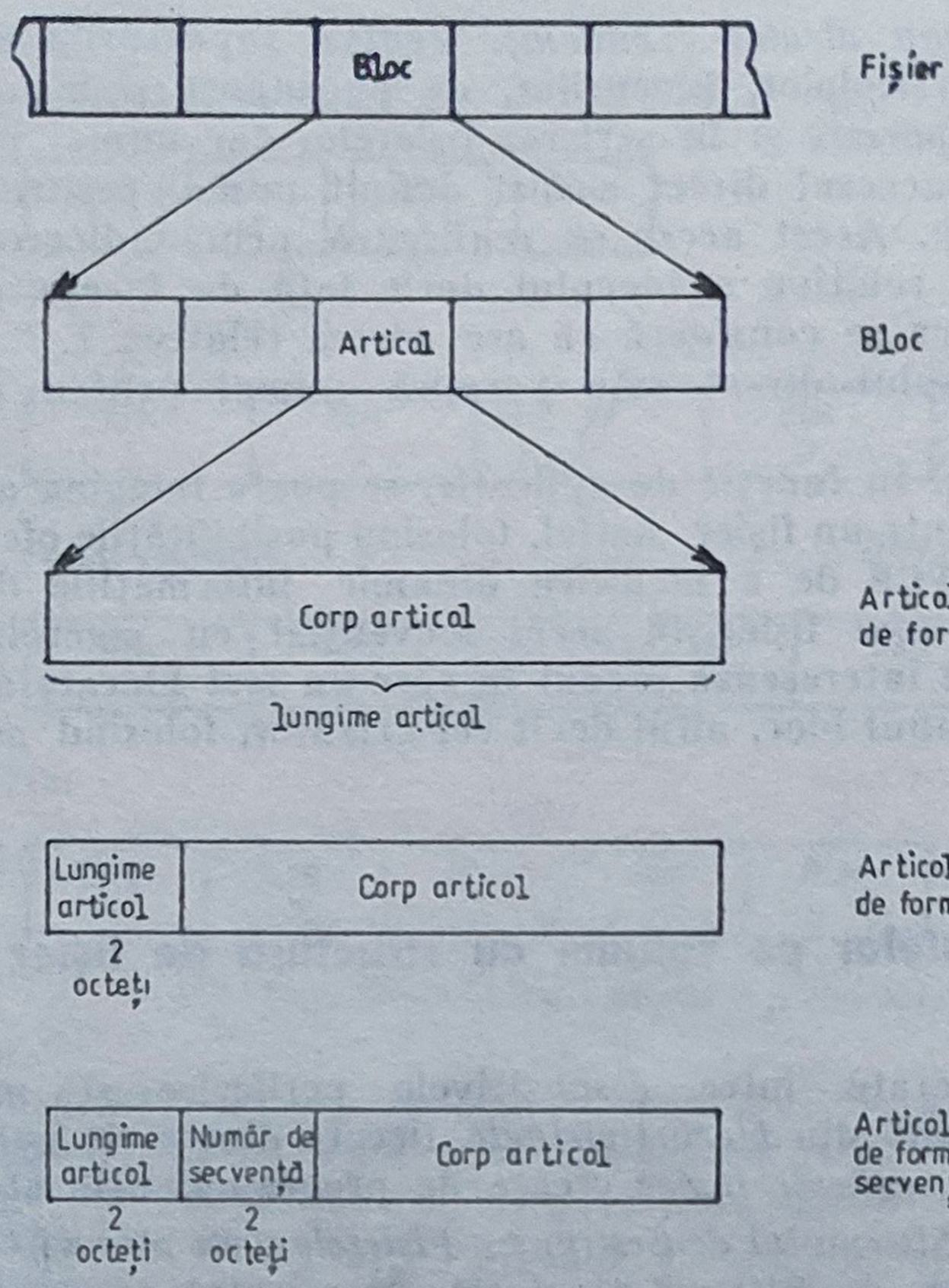


Fig. 12.3. Formatul articolelor într-un fișier MIX

Articolele de lungime fixă sînt împachetate la nivel de bloc fără informații de control sau caractere de umplere pentru aliniere. Blocul este trunchiat după sfîrșitul ultimului articol din buferul de bloc.

Înregistrările de lungime variabilă sînt precedate, pe bandă magnetică, de un contor de caractere (pe 4 octeți), sub forma unor caractere ASCII zecimale. Contorul include lungimea înregistrării și cei 4 octeți de contor. După ultima înregistrare din bloc, dacă există încă spațiu liber în acesta, sfîrșitul informațiilor din bloc este semnalizat de caracterul (^, cod **ASCII** 136).

12.5. Operații de intrare/ieșire asupra fișierelor

Sistemul de Gestiune a Fișierelor **FCS** asistă programele utilizator în efectuarea operațiilor de intrare/ieșire orientate pe înregistrare (articol) sau bloc și execută funcții adiționale cerute de controlul fișierelor, cum ar fi: deschidere, închidere, așteptare, ștergere etc. Sistemul de Gestiune a Fișierelor **FCS** asigură de asemenea blocarea și deblocarea articolelor, gestiunea zonelor tampon etc.

12.5.1. Operații de I/E la nivel de bloc

Pentru a se putea citi sau scrie blocurile virtuale de date, fără a ține cont de înregistrările logice din fișier, Sistemul de Gestiune a Fișierelor **FCS** pune la dispoziția utilizatorului macroinstrucțiunile **READ$** și **WRITE$.**

Acest mod de lucru reprezintă un mijloc foarte eficient de prelucrare a fișierelor deoarece el nu presupune blocarea/deblocarea articolelor.

Numărul blocului virtual va fi transmis, ca parametru, macroinstrucțiunii **READ$** și **WRITE$** ; blocul virtual astfel precizat va fi scris/citit din/în buferul rezervat, în acest scop, de către utilizator în programul său. Controlul asupra felului în care s-a terminat operația de I/E și sincronizarea operațiilor de I/E lansate astfel se face de către utilizator, fie prin utilizarea macroinstrucțiunii **WAIT$** (specificînd un indicator de eveniment folosit ca parametru de apel de macroinstrucțiunile **READ$/WRITE$)**, fie prin utilizarea unei rutine de tratare a întreruperilor asincrone **(AST)** (a cărei adresă va fi transmisă, ca parametru de apel, macroinstrucțiunilor **READ$/WRITE$)**.

12.5.2. Operații de I/E la nivel de articol

Sistemul de Gestiune a Fișierelor **FCS** pune la dispoziția utilizatorilor macroinstrucțiunile **GET$** și **PUT$** pentru prelucrarea fișierelor la nivel de articol.

Utilizînd buferele din zona de reentranță **FSR,** aceste macroinstrucțiuni efectuează operațiile necesare de blocare/deblocare a articolelor dintr-un bloc virtual al fișierului, permițînd astfel utillzatorului citirea sau scrierea unor articole logice.

Spre deosebire de operațiile de I/E la nivel de bloc, toate operațiile de I/E la nivel de articol sînt sincronizate ; controlul este redat programului utilizator numai după ce cererea de I/E a fost completată.

Atunci cînd se utilizează operațiile de I/E la nivel de articol, un program poate avea acces la articol în *două moduri* (după ce blocul a fost transferat în zona **FSR** dintr-un fișier) :

1. În modul **MOVE,** adică articolele vor fi transferate, din buferul ce se găsește în zona **FSR,** într-un bufer de articol rezervat în acest scop de către utilizator ;
2. În modul **LOCATE,** adică utilizatorul va dispune, într-o locație din **FDB,** de adresa articolului din buferul ce se găsește în zona **FSR.**

Modul **MOVE.** Acest mod presupune ca articolele să fie mutate între buferul din **FSR** și buferul de articol rezervat în programul utilizator. În cazul citirii **(GET$),** se citește mai întîi o înregistrare fizică în buferul din **FSR** și apoi fiecare articol, în funcție de operația de deblocare, este transferat în buferul din programul utilizator. În cazul scrierii **(PUT$),** utilizatorul va construi articolul în buferul de articol din programul său și apoi va muta articolul în buferul din zona **FSR,** iar cînd întreaga înregistrare fizică a fost umplută prin operația de blocare, ea va fi transmisă dispozitivului de I/E.

Modul de transfer **MOVE** simulează citirea unui articol direct în buferul din programul utilizator făcînd ca operațiile de blocare și deblocare a articolelor să fie transparente utilizatorului.

Modul **LOCATE.** Modul **LOCATE** dă posibilitatea utilizatorului de a avea acces la articol direct în buferul din zona **FSR.** Pentru aceasta este necesar ca utilizatorul să folosească anumite locații din blocul **FDB,** locații ce conțin valori care definesc lungimea și adresa articolului cerut. Prezența unui bufer de articol în programul utilizator nu este necesară decît în cazul în care articolele nu se aliniază, prin operația de blocare, la limita de bloc.

Dacă se dorește scrierea articolelor în modul **LOCATE** atunci rutinele **FCS** vor comunica utilizatorului, înainte de lansarea operației de scriere, adresa de memorie unde el își va putea construi articolul. Rămîne în responsabilitatea utilizatorului să indice lungimea articolului descris.

De asemenea, în lucrul cu fișiere la nivel de articol, cu ajutorul macroinstrucțiunii **FSRSZ$,** utilizatorul va trebui să rezerve o zonă de memorie pentru ca rutinele **FCS** să poată efectua operațiile de blocare/deblocare a articolelor în această zonă.

Utilizarea buferelor multiple și a buferelor mari

În desfășurarea operațiilor de I/E la nivel de articol, Sistemul de Gestiune a Fișierelor **FCS** utilizează bufere multiple, ce permit citirea datelor cu anticipație față de solicitările de acces utilizator și scrierea conținutului bufeleror în paralel cu construirea articolelor de ieșire. Aceasta permite suprapunerea prelucrărilor interne de date cu operațiile de I/E la nivel de fișier. Utilizarea buferelor multiple este avantajoasă în accesul secvențial al articolelor dintr-un fișier ; pentru accesul aleator *(random),* prelucrarea în paralel poate fi în unele cazuri ineficientă. Spațiul necesar buferelor multiple se alocă în programul utilizator cu macroinstrucțiunea **FSRSZ$.**

În cazul în care fișierele de date conțin articole de lungime mare sau grupări de articole *(cluster),* se folosesc bufere mari. Utilizarea acestei facilități reduce numărul de accese disc și permite intrări/ieșiri simultane pentru mai multe blocuri în secvență. Pentru accesul aleator *(random)* este necesar să se specifice un bufer utilizator. Spațiul necesar buferelor mari se alocă în programul utilizator cu macroinstrucțiunea **FSRSZ$,** ce precizează dimensiunea buferelor ca multiplu de 512 octeți.

În sistemul de operare **MIX-PLUS,** modulele de acces **FCS** și biblioteca rezidentă **FCSANS** includ implicit suportul necesar pentru bufere multiple și bufere mari.

12.5.3. Partajarea accesului la fișiere

Pentru fișierele de date plasate pe discuri magnetice, Sistemul de Gestiune a Fișierelor permite accesul multiplu, în concordanță cu anumite convenții. Aceste convenții se referă la deschiderea fișierelor utilizînd macroinstrucțiunile **OPNS$x** și **OPEN$x.**

1. Atunci cînd se folosește macroinstrucțiunea **OPNS$R** citirea fișierelor este totdeauna posibilă. O cerere de acces în scriere pentru același fișier va fi onorată numai și numai dacă o singură cerere de acest fel este făcută la un moment dat. Astfel pentru un fișier pot exista, la un moment dat, mai

multe cereri de citire și numai o singură cerere de scriere/creare (**W**), punere la zi (**U**), modificare (**M**) sau adăugare (**A**)).

2. Atunci cînd se folosește macroinstrucțiunea **OPEN$R** este permis accesul la fișier în citire, pentru mai mulți utilizatori, și nu este permis nici un fel de acces în scriere.

Accesul multiplu în citire nu implică în mod necesar ca cererile de acces să fie făcute de task-uri separate. Același task poate deschide același fișier utilizînd numere logice de dispozitiv diferite.

3. Atunci cînd se folosesc macroinstrucțiunile **OPNS$W**, **OPNS$M**, **OPNS$U** și **OPNS$A**, ce permit accesul multiplu la fișier în scriere, sincronizarea accesului la fișier revine task-ului apelant. Pentru eliminarea unor situații de distrugere multiplă a unor fișiere este necesară implementarea unui mecanism propriu fiecărui task pentru serializarea acceselor la același fișier.

În tabela 12.1 sînt sintetizate cazurile de acces multiplu ale unui fișier deschis în acces partajat.

Tabela 12.1

Modalitățile de acces multiplu ale unui fișier deschis în acces partajat

Acces secundar	Acces primar			
	Citire	Citire partajată	Scriere	Scriere partajată
Citire	Da	Da	Nu	Nu
Citire partajată	Da	Da	Da	Da
Scriere	Nu	Da	Nu	Nu
Scriere partajată	Nu	Da	Nu	Da

12.6. Structuri de date FCS

Pentru a putea exploata fișiere de date, cu ajutorul Sistemului de Gestiune a Fișierelor, un utilizator trebuie să asigure în programul său existența unor structuri de date care să specifice rutinelor de acces **FCS** tipul prelucrării dorite, identificatorii de fișier, adresa zonelor tampon etc. Aceste structuri de date sînt descrise în cele ce urmează.

12.6.1. Blocul de descriere a fișierului

Pentru a putea realiza funcțiile de deschidere, închidere, ștergere etc. de fișiere, precum și pentru lansarea operațiilor de I/E la nivel de bloc sau la nivel de articol, modulele de acces **FCS** utilizează informațiile înscrise în blocul de descriere a fișierului (**FDB**). Un astfel de bloc conține cinci secțiuni (vezi și fig. 12.4) :

1. Secțiunea *„atribute fișier“* ;
2. Secțiunea *„acces la articol“* sau *„acces la bloc“* ;

SECȚIUNEA „ATRIBUTE FIȘIER"

Formatul articolelor (tip, atribute)
Lungimea articolelor
Dacă articolele depășesc sau nu limita de bloc
Adresa ultimului bloc de fișier

SECȚIUNEA „ACCES LA ARTICOL"

Modul de acces la articol, lungime
Adresa buffer-ului utilizator pentru articol
Adresa și lungimea articolului citit / scris
Adresa următorului articol
Etc.

SECȚIUNEA „ACCES LA BLOC"

Bloc de stare al I/E (IOSB)
Adresa rutinei AST
Adresa și lungimea buffer-ului

SECȚIUNEA „DESCHIDERE FIȘIER"

Tipul de acces la fișier
Adresa structurilor de date ce definesc specificatorii de fișier
Numărul logic al perifericului

SECȚIUNEA „ZONE TAMPON FIȘIER"

Numărul indicatorului de eveniment utilizat în operațiile de I/E
Codul de eroare
Lungime buffer
Adresa curentă a blocului pe suport, etc.

SECȚIUNEA „NUME DE FIȘIER"

Identificatorul de fișier
Nume de fișier
Tip fișier
Versiune fișier
Codul de identificare al utilizatorului
Numele și numărul perifericului

Fig. 12.4. Structura blocului de descriere a fișierelor (FDB)

3. Secțiunea de *„deschidere fișier"* ;
4. Secțiunea de *„zone tampon"* ;
5. Secțiunea de *„nume de fișier"*.

Completarea și modificarea informațiilor în **FDB** se poate face cu ajutorul macroinstrucțiunilor **FCS.** Informațiile din **FDB** vor fi folosite, în mo-

mentul execuției, de către rutinele de acces **FCS** pentru a reliza acțiunea cerută de utilizator. Completarea eronată a **FDB**-ului sau alterarea unor informații în timpul execuției poate duce la executare incorectă a funcțiilor **FCS.**

12.6.2. Blocul de identificare implicită a fișierului și descriptorului de fișier

Pentru căutarea/crearea unui fișier pe un suport, rutinele **FCS** utilizează informațiile de identificare a fișierului din secțiunea „nume de fișier" din **FDB.** Spre deosebire de celelalte secțiuni ale **FDB**-ului, această secțiune este inițializată în timpul execuției programului de către rutinele **FCS.** Informațiile conținute în această secțiune sînt preluate din blocul de identificare implicită a fișiereului și/sau descriptorului de fișier.

a) *Descriptorul de fișier* **(DSD)**

Această structură de date cuprinde 6 cuvinte de memorie cu ajutorul cărora utilizatorul indică adresa unor șiruri de caractere care definesc identificatorii de fișier (nume fișier, cod de identificare al utilizatorului, nume dispozitiv de I/E etc. ...). La completarea secțiunii „nume de fișier" din **FDB,** rutinele **FCS** caută, mai întîi, informațiile de identificare a fișierului în descriptorul de fișier. În cazul în care un utilizator nu a prevăzut această structură de date se trece mai departe, la căutarea identificatorilor de fișier în blocul de identificare implicită a fișierului.

b) *Blocul de identificare implicită a fișierului* **(DFNB)**

Blocul de identificare implicită a fișierului este o structură de date care conține informații identice cu cele din secțiunea nume de fișier din **FDB.** Acest bloc poate fi rezervat de către utilizator prin folosirea macroinstrucțiunii **NMBLK$.** Informațiile din acest bloc sînt folosite pentru inițializarea secțiunii „nume de fișier" din **FDB,** dacă nu a fost prevăzut un descriptor de fișier, sau sînt folosite pentru completarea secțiunii „nume de fișier" din **FDB** cu informații care nu au fost transmise rutinelor **FCS** prin intermediul descriptorului de fișier.

12.6.3. Zonele tampon FCS și zona de reentranță FCS

Cu ajutorul macroinstrucțiunii **FSRZSZ$** utilizatorul va rezerva zonele tampon **FCS** și o zonă de reentranță **FCS.**

Zona tapon **FCS, $$FSR1,** conține o zonă de memorie necesară în operațiile de blocare/deblocare la articole, atunci cînd se lucrează la nivel de articol. Dacă se lucrează la nivel de bloc, această zonă **FCS** nu este necesară.

Zona de reentranță **FCS, $$FSR2,** este o zonă de memorie în care sînt memorate diverse date necesare rutinelor **FCS** în lansarea și controlul terminării operațiilor de I/E.

12.6.4. Zona tampon de articol

Atunci cînd se lucrează la nivel de articol, iar transferul articolelor se face în **MOVE** sau **LOCATE**, în general se va rezerva o zonă tampon de articol în programul utilizator. Adresa acestei zone tampon va fi indicată rutinelor **FCS** prin intermediul macroinstrucțiunilor **FCS**.

În modul **MOVE**, în citire, rutinele **FCS** efectuează operațiile de deblocare a articolelor din zonele tampon **FCS**, iar articolul citit se transferă în zona tampon de articol. La scrierea articolelor, rutinele **FCS** vor prelua articolul din zona tampon de articol și vor realiza operația de blocare a articolelor în zona tampon **FCS**.

În modul **LOCATE**, zona tampon de articol se folosește de către rutinele **FCS** numai în cazul articolelor ce depășesc limita de bloc.

12.7. Interfețe de apel FCS

Utilizatorii sistemului de operare **MIX** au la dispoziție un set de macroinstrucțiuni cu ajutorul cărora pot apela serviciile sistemului de gestiune a fișierelor **FCS**, pentru prelucrarea fișierelor de date. Aceste macroinstrucțiuni permit selectarea rutinelor **FCS** specifice aplicației dorite, numindu-se macroinstrucțiuni **FCS**.

Macroinstrucțiunile **FCS** ajută utilizatorii în :

— definirea și rezervarea unui bloc **FDB** ;
— definirea și rezervarea unui bloc de identificare implicită a fișierelor ;
— definirea și rezervarea zonelor de bufere **FCS** și a zonei de reentranță **FCS** ;
— completarea și modificarea informațiilor din **FDB** ;
— definirea unor simboli specifici ;
— apelul de rutine **FCS**.

12.7.1. Macroinstrucțiuni de descriere a structurilor de date

Macroinstrucțiunile **FCS** care acționează asupra blocului de descriere a fișierului (**FDB**) se pot împărți în :

— macroinstrucțiuni de definire și rezervare **FDB** ;
— macroinstrucțiuni de completare **FDB**, în timpul asamblării ;
— macroinstrucțiuni de completare și modificare **FDB**, în timpul execuției.

Din prima categorie fac parte macroinstrucțiunile :

— **FDBDF$** de rezervare a unui bloc **FDB** ;
— **FDOFF$** de definire a deplasărilor relative în cadrul unui bloc **FDB**.

Din a doua și a treia categorie fac parte macroinstrucțiunile :

FDAT$*x* — pentru completarea și/sau modificarea secțiunii de „atribute articol" din **FDB**. Această macroinstrucțiune se folosește numai atunci cînd se dorește crearea unui nou fișier ;

FDRC$x — pentru completarea și/sau modificarea secțiunii de „acces la articol“ din **FDB.** Cu ajutorul acestei macroinstrucțiuni utilizatorul va specifica adresa și lungimea zonei tampon de articol ;

FDBK$x — pentru completarea și/sau modificarea secțiunii de „acces la bloc“ din **FDB.** Cu ajutorul macroinstrucțiunii utilizatorului va indica adresa și lungimea unei zone tampon în care se vor citi sau din care se vor scrie blocurile (de) pe suport. De asemenea se mai poate indica cu ajutorul acestei macroinstrucțiuni și adresa unui bloc al stării operației de I/E lansate **(IOSB)** sau adresa unei rutine de prelucrare a întreruperilor asincrone **(AST)** ;

FDOP$x — pentru completarea și/sau modificarea secțiunii de „deschidere fișier“ din **FDB.** Cu ajutorul macroinstrucțiunii, utilizatorul poate indica numărul logic al dispozitivului de I/E, tipul de prelucrare dorit (creare fișier, citire fișier etc. ...), adresa altor informații privind dispozitivul de I/E (rebobinare, poziționare pe sfîșit de volum etc.) ;

FDBF$x — pentru completarea și sau modificarea secțiunii „zone tampon **FCS**“ din **FDB.** Numărul evenimentului semnificativ asociat operațiilor de I/E, eventual, lungimea blocului de scriere/citire se pot indica cu ajutorul acestei macroinstrucțiuni.

Semnificația particulei x este următoarea :

A — pentru macroinstrucțiunile folosite la completarea **FDB,** în timpul asamblării ;

R — pentru macroinstrucțiunile folosite la completarea **FDB,** în timpul execuției.

Atunci cînd aceste macroinstrucțiuni sînt folosite pentru completarea **FDB** în timpul asamblării programului, ele trebuie să urmeze obligatoriu după macroinstrucțiunea de rezervare a blocului **FDBFDBDF$.** Argumentele de apel ale macroinstrucțiunilor de acest tip trebuie să fie expresii sau numere recunoscute de către limbajul **MACRO** ca operanzi valizi ai directivelor **.BYTE** și **.WORD.**

Pentru completarea și/sau modificarea informațiilor din **FDB** se pot utiliza aceleași macroinstrucțiuni. Spre deosebire de macroinstrucțiunile de completare a **FDB** în timpul asamblării, macroinstrucțiunile de completare/modificare a **FDB** în timpul execuției se vor folosi astfel :

— oriunde în program unde se dorește modificarea/completarea **FDB** în timpul execuției ;

— argumentele macroinstrucțiunii trebuie să fie expresii sau numere recunoscute de limbajul **MACRO** drept operanzi valizi ai instrucțiunii **MOV.**

Pentru definirea și rezervarea blocului de identificare implicită a fișierului, utilizatorul dispune de macroinstrucțiunile **FCS :**

NMBLK$ — pentru rezervarea și completarea blocului de identificare implicită a fișierelor ;

NBOFF$ — pentru definirea deplasărilor relative la începutul blocului.

Zonele tampon **FCS** și zona de reentranță **FCS** pot fi definite, rezervate și completate cu ajutorul macroinstrucțiunilor :

FSRSZ$ — pentru rezervarea zonelor tampon **FCS** și a zonei de reentranță **FCS ;**

FSROF$ — pentru definirea deplasărilor relative în zona de reentranță **FCS** ;
FINIT$ — pentru completarea/modificarea, în timpul execuției, a datelor din zona de reentranță **FCS.**

12.7.2. Macroinstrucțiunile de apel ale funcțiilor FCS

În timpul execuției unui program, blocul **FDB** poate fi modificat și cu ajutorul macroinstrucțiunilor de apel ale funcțiilor **FCS :**
OPEN$*x* — pentru deschiderea unui fișier (caz general) ;
OPNS$*x* — pentru deschiderea unui fișier în acces multiplu (partajat) ;
OFID$*x* — pentru deschiderea unui fișier existent, în funcție de identificatorul de fișier din **FDB** ;
OFNB$*x* — pentru deschiderea unui fișier în funcție de numele său (din blocul de identificare implicită).
Valorile *x* pot fi :
blank — procedura generală de deschidere ;
R — citire fișier existent ;
W — scriere (creare) nou fișier ;
M — modificarea fișier existent fără modificarea lungimii ;
U — actualizarea fișier existent și extinderea lungimii ;
A — adăugare informații la sfîrșitul unui fișier existent.
OPNT$*y* — pentru crearea și deschiderea unui fișier de manevră ;
Valorile *y* pot fi :
W — deschidere și creare fișier temporar pe durată limitată ;
D — deschidere și creare fișier temporar și marcare pentru ștergere.
CLOSE$ — pentru închiderea fișierului ;
GET$*z* — pentru citirea articolelor unui fișier.
Valorile *z* pot fi :
blank — procedura generală de citire ;
R — citire articole de lungime fixă dintr-un fișier cu acces direct ;
S — citire articole dintr-un fișier cu acces secvențial.
PRINT$ — pentru tipărirea asincronă a conținutului unui fișier ;
PUT$*x* — pentru scrierea articolelor într-un fișier.
Valorile *x* pot fi :
blank — procedura generală de scriere ;
R — scriere articole de lungime fixă într-un fișier cu acces direct ;
S — scriere articole într-un fișier cu acces secvențial.
READ$ — pentru citirea unor blocuri virtuale dintr-un fișier de organizare nedefinită ;
WRITE$ — pentru scrierea unor blocuri virtuale într-un fișier de organizare nedefinită ;
DELET$ — pentru ștergerea unui fișier dintr-un catalog asociat unui volum disc, cu dealocarea spațiului ocupat ;
WAIT$ — pentru suspendarea execuției programului pînă la terminarea operației de I/E solicitate, la nivel de bloc.

Execuția acestor macroinstrucțiuni poate fi testată de către utilizator în funcție de codurile de retur ale rutinelor **FCS.**

Executarea corectă a unei macroinstrucţiuni va fi semnalată de către rutinele **FCS** prin poziţionarea pe 0 a bitului de condiţie „C" din Starea Program (**PS**). Corespunzător, într-o locaţie din **FDB** se va găsi un cod de retur cu o valoare pozitivă.

Un cod de retur negativ şi bitul „C" poziţionat pe 1 va indica utilizatorului o execuţie incorectă a macroinstrucţiunii. Erorile **FCS** şi erorile detectate de către driverele dispozitivelor de I/E pot fi recunoscute prin testarea locaţiei din **FDB** imediat următoare codului de retur **FCS.** O valoare pozitivă sau 0 în această locaţie va indica o eroare **FCS** sau o eroare detectată de driver. O valoare negativă în această locaţie va fi o indicaţie că eroarea detectată este o eroare de directivă Monitor, respectiv că parametrii transmişi unei directive Monitor au fost incorecţi.

Pentru a uşura accesul utilizatorului la structura unor blocuri **FCS** (**FDB, $$FSR2,** blocul de identificare implicită a fişierelor etc.) au fost definite o serie de valori simbolice standard. Aceste valori simbolice definesc deplasarea relativă a unei locaţii, în cadrul unei structuri de date **FCS,** sau codul de retur **FCS,** după executarea unei macroinstrucţiuni. Deplasările relative în cadrul unei structuri de sate **FCS** pot fi definite ca simboluri globale sau locale utilizînd macroinstrucţiunile **FCS : FDOFF$, NBOFF$** etc.

În cap. 30 (vol. 2) se prezintă modul de utilizare a macroinstrucţiunilor **FCS** în scrierea programelor în limbajul **MACRO.**

12.8. Rutine cu funcţii specifice FCS

Utilizatorii cu o experienţă mare în lucrul cu fişiere şi care cunosc bine posibilităţile Sistemului de Gestiune a Fişierelor **FCS,** pot folosi o serie de rutine **FCS** pentru a realiza funcţii specifice **FCS** :

.ASCPP — rutina asigură conversia unui şir de identificare catalog utilizator din **ASCII** în binar ;

.ASLUN — rutina asigură asignarea unui dispozitiv de I/E la număr logic ;

.CTRL — rutina permite efectuarea unor operaţii specifice unei unităţi de dispozitiv, în special pentru banda magnetică (rebobinare, poziţionare pe sfîrşitul logic al unui volum etc.) ;

.DLFNB — rutina asigură ştergerea unui fişier de pe un suport, în funcţie de numele fişierului prezent în **FDB** ;

.ENTER — rutina asigură crearea unui nume de fişier într-un catalog, eventual cu rezervarea spaţiului pe suport ;

.EXTND — rutina asigură extensia spaţiului ocupat de un fişier pe suport, cu un număr de blocuri specificat de utilizator ;

.FIND — rutina permite localizarea unui fişier pe suport şi completarea cîmpului de identificare fişier din **FDB** ;

.FLUSH — rutina forţează scrierea unui bufer de bloc în fişierul specificat ;

.GTDID — rutina asigură căutarea unui fişier pe suport în funcţie de un şir de identificare catalog utilizator stabilit implicit de **FCS** (în secţiunea de program **$$FSR2**) ;

.GTDIR — rutina permite căutarea unui fișier pe suport în funcție de un șir de identificare catalog utilizator și completarea blocului de identificare a fișierului ;

.MARK — rutina permite salvarea contextului actual al unui fișier (poziția curentă de acces) ;

.MRKDL — rutina permite luarea în evidență a unui fișier care urmează să fie șters de pe suport ;

.PARSE — rutina analizează structurile de date **FCS** pentru completarea **FDB** cu informațiile necesare identificării unui fișier ;

.POINT — rutina permite poziționarea unui fișier pe un octet precizat al unui articol dintr-un fișier care a fost în prealabil deschis. Se utilizează foarte des după ce s-a executat, cu rutina **.MARK**, o salvare a contextului actual al unui fișier ;

.POSIT — rutina calculează numărul de bloc virtual și deplasarea în bloc pentru un articol specificat ;

.POSRC — rutina forțează poziționarea în fișier pe o înregistrare de lungime fixă specificată și pregătește **FDB** pentru operația **PUT** în modul **LOCATE** ;

.PPASC — rutina asigură conversia unui șir de identificare catalog utilizator din binar în **ASCII** ;

.PRSDV — rutina permite analiza și ulterior completarea **FDB** cu numele și numărul de unitate al dispozitivului de I/E pe care utilizatorul dorește să-l utilizeze în programul său ;

.PRSDI — rutina analizează structurile **FCS** și completează **FDB** cu informații de identificare a catalogului de fișier ;

.PRSFN — rutina analizează structurile **FCS** și completează **FDB** cu informații de identificare a numelui, tipului și versiunii de fișier ;

.RDFDR — rutina permite citirea, din zona de reentranță **FCS (\$\$FSR2)**, a lungimii și adresei șirului de identificare a catalogului utilizator ;

.RDFFP — rutina permite citirea cuvîntului de protecție al fișierului din zona de reentranță **FCS (\$\$FSR2)** ;

.RDFUI — rutina permite citirea codului de identificare utilizator implicit din zona de reentranță **FCS (\$\$FSR2)** ;

.REMOV — rutina elimină (șterge) o intrare dintr-un catalog pentru un nume de fișier specificat ;

.RENAM — rutina asigură schimbarea numelui de fișier în catalogul asociat ;

.RFOWN — rutina citește și returnează conținutul specificatorului de proprietar fișier din secțiunea program a **FSR (\$\$FSR2)** ;

.TRNCL — rutina trunchiază un fișier în punctul de sfîrșit logic, dezalocă spațiul după sfîrșitul de fișier logic și închide fișierul ;

.WDFDR — rutina permite modificarea adresei și lungimii șirului de identificare a catalogului utilizator, din zona de reentranță **FCS (\$\$FSR2)** ;

.WDFFP — rutina permite modificarea cuvîntului de protecție al fișierului din zona de reentranță **FCS(SSFSR2)** ;

.WDFUI — rutina permite scrierea unui nou cod de identificare utilizator implicit în zona de reentranță **FCS(SSFSR2)** ;

.WFOWN — rutina inițializează conținutul specificatorului de proprietar fișier din secțiunea program a **FSR(SSFSR2)** ;

.XQIO — rutina permite utilizatorului lansarea și completarea unor operații de I/E specifice **FCS.**

O execuție corectă a acestor rutine va fi semnalată prin poziționarea pe 0 a bitului "C" din starea program și printr-un cod de retur **FCS** cu o valoare pozitivă sau 0. Dacă bitul "C" este egal cu 1 și codul de retur **FCS** este negativ, atunci utilizatorul va ști că rutina nu s-a executat corect și în funcție de codul de retur **FCS**, poate să ia o decizie în conformitate cu aplicația pentru care a fost scris programul.

Rutinele de completare a **FDB** cu informațiile de identificare a fișierului specificat **(.PARSE, .PRSDV, .PRSDI, .PRSFN, .ASLUN)** suportă în mod standard, *nume logice extinse* îm sensul **MIX-PLUS** (a se vedea precizările aduse în cap. 9, paragraful 2,2 și cap. 11, paragraful 9). Rutinele de acces **FCS** sînt însă unice pe sistemele de operare **MIX** și **MIX-PLUS.** În cazul folosirii unor nume logice diferite de formatul **ddnn :**, sub sistemul de operare **MIX** se va semnaliza eroare. Pe sistemul de operare **MIX-PLUS** se permite orice combinație de nume logice în specificatorul de fișier.

12.9. Biblioteci rezidente FCS

Sistemul de Gestiune a Fișierelor **FCS** se poate implementa astfel :

— ca module de acces atașate, la editarea de legături, programului utilizator ;

— ca bibliotecă rezidentă de rutine partajate ;

— dinamic, prin combinarea primelor două posibilități.

În primul caz, la editarea de legături a unui program, se vor selecta din biblioteca sistem de module obiect, cu specificatorul **LB : [1, 1] SYSOLIB .OLB,** acele module **FCS** pe care utilizatorul dorește să le folosească în timpul prelucrării datelor.

În cel de-al doilea caz, se folosește proprietatra modulelor **FCS** de a fi reentrante.

Sistemele de operare **MIX/MIX-PLUS** conțin în mod standard o bibliotecă de rutine partajate, rezidentă, cu specificatorul de fișier **LB : [1,1] FCSANS .TSK.**

În sistemul de operare **MIX-PLUS,** biblioteca modulelor de acces **FCS** poate fi implementată și ca o bibliotecă mapată în modul *Supervizor,* pentru a extinde spațiul virtual atașat task-urilor utilizator. Specificatorul de fișier al acestei biblioteci este **LB : [1, 1] FCSFSL.TSK.**

Ambele biblioteci sînt implementate sub forma unor structuri segmentate rezidente în memorie, ce se mapează la programele utilizator printr-un **APR** (4 Kcuvinte). Programele utilizator se pot lega în mod static sau dinamic la aceste biblioteci partajate ce conțin totalitatea serviciilor de acces **FCS.**

Utilitarele şi compilatoarele livrate standard sub sistemele de operare **MIX/MIX-PLUS** beneficiază de această facilitate, ducînd la creşterea performanţelor de exploatare şi acces la structurile de fişiere utilizator.

În sistemul de operare **MIX-PLUS** este posibilă şi utilizarea unei biblioteci unice de rutine de acces **FCS**, mapate în modul de lucru *Utilizator* sau *Supervizor*, în funcţie de spaţiul virtual de acces al task-urilor apelante.

Realizarea acestui deziderat, ce reduce spaţiul disc şi de memorie asociat, este posibilă prin următoarle acţiuni:

— includerea în biblioteca cu specificatorul **LB : [1, 1] FCSANS.TSK** a unor rutine (de completare) specifice modului *Supervizor ;*
— maparea diferenţiată a bibliotecii de acces la editarea de legături:
  — pentru maparea în modul *Utilizator*, se link-editează task-urile utilizator cu opţiunea **LIBR = FCSANS** (1 APR *Utilizator* de acces) ;
  — pentru maparea în modul *Supervizor*, se link-editează task-urile utilizator cu opţiunea **SUPLIB = FCSANS** (1 **APR** *Utilizator* şi 1 **APR** *Supervizor* de acces).

Utilizarea acestui mod dual de acces este posibilă numai pentru sistemele **MIX-PLUS** cu spaţii **I** şi **D** separate, funcţionale pe minicalculatoarele româneşti **I-106** şi **I-1016.**

13

Sistemul de gestiune a înregistrărilor (RMS)

13.1. Prezentare generală

Sistemul de gestiune a înregistrărilor (**RMS** — *Record Management Services*) asigură un set de servicii și facilități extrem de importante pentru gestiunea informațiilor, punînd la dispoziția utilizatorilor metode eficiente și flexibile de memorare și regăsire a unităților de date logice (înregistrări) pe volume disc.

Accesul la sistemul de gestiune a înregistrărilor **RMS** este posibil din limbajul **MACRO**, prin apeluri de macroinstrucțiuni, specifice, sau dintr-un limbaj de nivel înalt, prin apeluri de rutine specifice.

Sistemul de gestiune a înregistrărilor controlează structura internă a fișierelor în funcție de :

— formatul înregistrărilor ;
— organizarea fișierelor ;
— modurile de acces la înregistrare ;
— operațiile efectuate asupra înregistrărilor ;
— operațiile efectuate asupra fișierelor.

13.2. Organizarea fișierelor RMS

La crearea unui fișier se poate alege unul din următoarele tipuri de organizare :

Organizarea secvențială

Ordinea fizică a înregistrărilor coincide cu ordinea în care s-au scris în fișier.

Organizarea relativă

Înregistrările se află în locații de lungime fixă numite celule, numerotate de la 1 la *n*.

Dimensiunea celulei reprezintă lungimea maximă a înregistrărilor din fişier, iar numărul ei, poziţia relativă faţă de începutul fişierului.

Fiecare celulă poate conţine o singură înregistrare dar poate fi şi vidă.

Organizarea indexată

Serviciile de acces **RMS** controlează aşezarea înregistrărilor în fişier după cheia aleasă de utilizator.

Cheia este un cîmp de înregistrare identificat prin adresă de început şi lungime.

Pentru fiecare fişier indexat, trebuie să se definească cel puţin o cheie — *cheia primară*. Opţional se pot defini *chei secundare*.

Valoarea cheii este folosită pentru identificarea articolului. **RMS** organizează valorile cheilor într-o tabelă cu structură de arbore numite index. Pentru fiecare cheie definită, **RMS** construieşte cîte un fişier index.

13.3. Moduri de acces la înregistrările fişierelor RMS

Metodele de regăsire şi memorare a articolelor în cadrul unui fişier sînt cunoscute sub denumirea de *moduri de acces*.

La fiecare deschidere de fişier se pot folosi moduri de acces diferite. Există de asemenea posibilitatea schimbării modului de acces în timpul prelucrării unui fişier, folosind o procedură numită *acces dinamic*.

RMS oferă *trei moduri de acces la articole:*

13.3.1. Accesul secvenţial

În *acces secvenţial*, înregistrările sînt găsite sau scrise în ordinea stabilită de organizarea de fişier.

Acest mod de acces poate fi folosit pentru toate fişierele **RMS** şi pentru toate perifericele orientate pe înregistrare.

Accesul secvenţial la fişiere secvenţiale

Ordinea în care sînt accesate înregistrările este ordinea fizică. Pentru găsirea unei anumite înregistrări dintr-un fişier secvenţial, este necesară accesarea tuturor înregistrărilor ce o preced fizic.

Într-o operaţie de scriere secvenţială, întotdeauna noua înregistrare urmează pe precedenta.

Accesul secvenţial la fişiere relative

Ordinea în care sînt accesate înregistrările este ordinea celulelor în care se află înregistrările.

Cînd se face o citire în mod secvenţial, într-un fişier relativ, se caută în celule succesive pînă se găseşte una care conţine o înregistrare.

Cînd un program adaugă înregistrări în mod secvenţial într-un fişier relativ, **RMS** respectă ordinea ascendentă a numerelor relative ale celulelor. Noua înregistrare se aşează în celula cu numărul relativ $n + 1$ (dacă aceasta este liberă), n fiind numărul celulei care a făcut obiectul accesului precedent.

Accesul secvențial la fișiere indexate

Ordinea în care sînt accesate înregistrările este ordinea indicată de index. Intrările în index sînt aranjate în ordinea ascendentă după cheie.

Dacă sînt definite mai multe chei, fiecare index asociat cheii reprezintă ordinea logică a înregistrărilor într-un fișier.

Cînd se citesc înregistrări dintr-un fișier indexat în mod secvențial, se specifică obligatoriu o cheie.

RMS folosește indexul asociat cheii respective și găsește astfel înregistrările în ordinea indicată de index.

Cînd se scrie secvențial într-un fișier indexat, nu e necesară specificarea unei chei. **RMS** folosește definiția cheii primare pentru a așeza articolele noi în fișiere.

13.3.2. **Accesul aleator**

În *acces aleator*, programul stabilește ordinea în care se prelucrează înregistrările. Fiecare cerere de acces la o înregistrare este tratată independent de cererea precedentă.

Cereri succesive în mod aleator pot identifica înregistrări oriunde în fișier.

Accesul aleator la fișiere secvențiale

Dacă un fișier secvențial se află pe disc și are înregistrări de lungime fixă, accesul aleator este posibil. Pentru localizarea înregistrării se folosește numărul relativ al înregistrării.

Accesul aleator la fișiere relative

Programele pot citi/scrie înregistrări în fișiere relative în mod aleator, specificînd numărul de articol relativ al înregistrării. **RMS** interpretează fiecare număr ca un număr corespunzător de celulă.

Într-o operație de citire este posibil ca celula specificată să nu conțină nici un articol. În acest caz, în program, se returnează un cod adecvat (articol inexistent).

Dacă într-o operație de scriere se indică un număr de celulă care este deja ocupată, **RMS** transmite programului codul de eroare — articol deja existent.

Acces aleator la fișierele indexate

Într-un fișier indexat înregistrarea este identificată după valoarea cheii.

Pentru a citi o înregistrare într-un fișier indexat, în modul de acces aleator, se specifică o valoare de cheie și un index (primar sau unul dintre cele secundare).

RMS localizează valoarea cheii în indexul specificat, apoi citește înregistrarea spre care adresează intrarea respectivă din index. Înregistrarea citită este dată programului apelant.

Scrierea în mod aleator într-un fișier indexat a unei noi înregistrări nu cere specificarea unei valori de cheie. Valorile tuturor cheilor se află în corpul înregistrării.

La deschiderea fișierului indexat, **RMS** se informează asupra definițiilor tuturor cheilor declarate. Înainte de scrierea efectivă a înregistrării, **RMS** examinează valorile conținute în cîmpurile de cheie ale respectivei înregistrări și creează noi intrări în indecși.

13.3.3. Accesul prin adresa înregistrării (RFA)

Acest mod de acces poate fi folosit pentru regăsirea înregistrărilor în fișiere cu orice tip de organizare, cu condiția ca aceste fișiere să se afle pe disc.

Se identifică înregistrarea după adresa sa unică (**RFA**), pentru o eventuală regăsire.

Formatul acestei adrese depinde de organizarea de fișier și numai **RMS** o poate interpreta.

După fiecare operație de citire/scriere, **RMS** returnează programului apelant adresa înregistrării. Utilizatorul o poate salva și folosi ulterior pentru regăsirea aceluiași articol (eventual la o nouă execuție a programului). Aceasta este o tehnică de optimizare folosită pe larg în programele utilizator.

13.3.4. Accesul dinamic

Accesul dinamic constă în posibilitatea de a schimba modul de acces în timpul prelucrării unui fișier. Numărul de comutări ale modului de acces nu este limitat. Singura limitare este ca organizarea de fișier să suporte modul de acces selectat.

*

În tabela 13.1 sînt sintetizate combinațiile permise între *tipurile de organizare a fișierelor* și *modurile de acces*.

Tabela 13.1

Combinații permise între modurile de acces și tipuri de organizare de fișiere

Organizarea de fișier	Mod de acces			
	Secvențial	Aleator		RFA
		Număr articol	Valoare cheie	
Secvențial	DA	NU	NU	DA
Relativ	DA	DA	NU	NU
Indexat	DA	NU	DA	DA

În tabela 13.2 sînt sintetizate combinațiile posibile între *tipurile de organizare a fișierelor* și *formatul înregistrărilor.*

Tabela 13.2

Formatul înregistrării și organizarea de fișier

Organizarea de fișier	Formatul înregistrării			
	Fix	Variabil	VFC	Șir
Secvențial	DA	DA	DA	numai pe disc
Relativ	DA	DA	DA	NU
Indexat	DA	DA	NU	NU

13.4. Interfețe de programare RMS

După crearea unui fișier **RMS**, un program utilizator îl poate accesa, citi/scrie și regăsi datele conținute în el.

Se pot folosi două modalități de acces la fișier :

1) Acces la nivel de bloc virtual, numit bloc de I/E. În acest caz, programul accesează fișierul ca o structură fizică. **RMS** consideră fișierul ca o zonă de blocuri virtuale ;

2) Acces la nivel de înregistrare. În acest caz, programul accesează fișierul ca o structură logică (fișier secvențial, relativ sau indexat). Operațiile de I/E folosite pot fi :

— adăugare înregistrări ;
— modificare înregistrări ;
— ștergere înregistrări.

Tipul operației permise asupra înregistrării depinde de organizarea de fișier.

13.4.1. Prelucrarea la nivel de fișier

Operațiile la nivel de fișier consideră fișierul ca o entitate, fără a se ocupa de înregistrările și blocurile ce-l compun.

RMS execută următoarele operații la nivel de fișier :

CREATE Creează fișier (cu o intrare corespunzătoare în catalog) și deschide fișierul pentru prelucrare ;
OPEN Deschide un fișier existent pentru prelucrare ;
DISPLAY Scrie informațiile despre fișier în blocuri de control ;
ERASE Șterge conținutul fișierului (înregistrări sau blocuri) și extrage intrarea corespunzătoare din catalog ;
EXTEND Mărește zona de memorie alocată fișierului ;
CLOSE Închide un fișier deschis.

RMS poate executa următoarele operații asupra intrărilor din cataloage, care nu afectează conținutul fișierelor :

ENTER Creează intrare în catalog ;
REMOVE Șterge intrare în catalog ;
RENAME Înlocuiește intrare în catalog ;
PARSE Analizează specificatorul de fișier ;
SEARCH Caută în cataloage.

13.4.2. Prelucrarea la nivel de înregistrare

RMS permite unui program să execute următoarele operații asupra înregistrărilor :

Citire înregistrare : — **RMS** returnează programului apelant o înregistrare din fișierul specificat ;

Scriere înregistrare : — **RMS** adaugă noua înregistrare construită de program fișierului specificat ;

Găsire înregistrare : — **RMS** localizează o înregistrare existentă într-un fișier. Nu returnează înregistrarea în program ci stabilește doar o nouă poziție curentă în fișier ;

Actualizare înregistrare : — Programul modifică conținutul unei înregistrări citite într-un fișier. **RMS** scrie înregistrarea modificată într-un fișier, înlocuind vechiul articol ;

Ștergere înregistrare : — **RMS** șterge o înregistrare dintr-un fișier. Operație interzisă într-un fișier secvențial.

Felul în care se fac operațiile la nivel de înregistrare depinde de tipul organizării de fișier, conform tabelei 13.3.

Tabela 13.3

Operațiile posibile la nivel de înregistrare

Operația la nivel înregistrare	Tipul organizării de fișier		
	Secvențială	Relativă	Indexată
1	2	3	4
Citire	Mod de acces : • secvențial • RFA	Mod de acces : • secvențial • aleator • RFA	Mod de acces : • secvențial • aleator : 1) cu potrivire exactă cheie ; 2) cu potrivire aproximativă cheie ; 3) cu potrivire generică cheie 4) cu potrivire aproximativă și generică cheie. • RFA
Scriere	Mod de acces : • secvențial	Mod de acces : • secvențial • aleator	Operație admisă. Restricție : Noua înregistrare să respecte caracteristicile cheii definite de utilizator.

Tabela 13.3 (continuare)

1	2	3	4
Găsire	Mod de acces : • secvențial • RFA	Mod de acces : • secvențial • aleator • RFA	Mod de acces : • secvențial • aleator — cu cele 4 tipuri de potrivire. • RFA
Actualizare	Operație admisă. Restricție : Lungimea înregistrării să nu se schimbe	Operație admisă. Se poate schimba și lungimea înregistrării (în cazul înregistrării de lungime variabilă)	Operație admisă. Restricție : Conținutul înregistrării modificate trebuie să respecte caracteristicile cheii definite de utilizator
Ștergere	—	Operație admisă. Celula înregistrării devenind disponibilă pt. o nouă înregistrare	Operație admisă

În cap. 31 (vol. 2) se prezintă detalii și exemple de utilizare în programare a diferitelor facilități **RMS**.

13.5. Facilități RMS pentru partajarea fișierelor

Fișierele **RMS** pot fi partajate între mai mulți utilizatori :

— cele secvențiale — partajate în citire ;

— cele relative și indexate — partajate în citire și scriere.

O serie de facilități **RMS** permit realizarea accesului concurent la fișierele **RMS**. Se vor prezenta aceste facilități corespunzător celor două nivele de prelucrare.

13.5.1. Facilități RMS pentru prelucrările la nivel de fișier

RMS și sistemul de operare MIX oferă un mediu de prelucrare care permite programelor, ce se execută concurent, să partajeze accesul la același fișier.

Felul în care se face această partajare depinde de tipul de organizare a fișierului partajat.

RMS utilizează facilitatea de blocare a paginii pentru a controla operațiile făcute în fișiere relative sau indexate partajate în scriere.,

Scopul acestei facilități este de a asigura ca un program să poată adăuga, șterge sau modifica înregistrări într-un fișier fără ca alt program să acceseze simultan aceeași înregistrare.

13.5.2. Facilităţi RMS pentru prelucrările la nivel de înregistrare

RMS permite programelor să acceseze înregistrările dintr-un fişier prin intermediul uneia sau mai multor căi de acces.

Fiecare cale de acces reprezintă o serie de cereri de operaţii care prelucrează o singură înregistrare la un moment dat.

Pentru fişierele relative şi indexate, **RMS** permite stabilirea mai multor căi de acces la înregistrările aceluiaşi fişier, asigurînd prelucrarea lor paralelă.

RMS foloseşte şi în acest caz mecanismul de blocare a paginii, pentru a controla partajarea fişierelor între diferite programe.

13.6. Interfeţe RMS cu acces la distanţă

Utilizînd reţeaua **MININET/MIX**, un program dintr-un nod poate accesa un fişier din alt nod (vezi şi cap. 14, vol. 1).

În contextul accesului la distanţă programul care cere acces se numeşte „*program-sursă*“ iar cel căruia i se adresează cererea se numeşte „*program-destinaţie*“.

Cînd un program sursă adresează o cerere de deschidere fişier la distanţă trebuie să se specifice : un specificator de fişier, informaţii de control acces şi caracteristicile fişierului ce urmează a fi accesat.

În specificatorul de fişier al programului destinaţie, există un cîmp în care se furnizează numele nodului. Astfel, după analiza liniei de comandă, **RMS** decide dacă se solicită acces la distanţă după cum cîmpul de nod e prezent sau nu în fişierul ce trebuie deschis.

În cap. 40 (vol. 2) se prezintă în detaliu, un exemplu de acces la distanţă într-o reţea, pentru fişiere **RMS**.

13.7. Configurarea de biblioteci de acces RMS rezidente

Începînd cu versiunea **MIX-PLUS V2.0**, în cadrul familiei de sisteme de operare **MIX/MIX-PLUS** a fost dezvoltată o nouă bibliotecă de module de acces **RMS**, caracterizată prin performanţe superioare şi *overhead* redus. Biblioteca este împărţită în trei segmente separate, care pot să nu fie fizic rezidente în memorie.

Partiţionarea permite solicitarea „la cerere“ a segmentelor de bibliotecă. La un moment dat vor fi rezidente în memorie numai acele segmente ce conţin codul utilizat. Restul segmentelor de bibliotecă sînt eligibile de a fi înlocuite (eliminate) din memorie de task-urile ce reclamă memorie fizică. Un task oarecare nu poate solicita simultan mai mult de două segmente de bibliotecă.

Funcţional, noua bibliotecă de acces **RMS** este configurată astfel :

— **RMSRES.TSK** (22 kocteţi) reprezintă „*rădăcina*“ bibliotecii segmentate (fişierul **.TSK** şi fişierul de simboluri. **STB** reprezintă minimum necesar

pentru legarea unui task la biblioteca **RMS**). Rădăcina **RMSRES** cuprinde funcțiile **$DISPLAY, $ERASE, $PARSE, $SEARCH, $RENAME, $FREE,** funcția comună internă **EXTEND**, toate operațiile asupra fișierelor secvențiale, toate operațiile de I/E la nivel de bloc și înregistrare, toate funcțiile asociate benzilor magnetice și o parte comună de cod pentru operațiile **$CREATE, $OPEN** și **$CLOSE** asociate fișierelor relative și indexate.

— **RMSLBL.TSK** (8 kocteți) reprezintă un *segment satelit* de bibliotecă ce suportă operații asupra fișierelor relative, funcția **$EXTEND** și operațiile **$OPEN, $CLOSE, $CONNECT, $DISCONNECT, $FIND** și **$GET** asupra fișierelor indexate.

— **RMSLBM.TSK** (14 kocteți) reprezintă un *segment satelit* de bibliotecă ce include restul operațiunilor asupra fișierelor indexate : **$DELETE, $PUT, $UPDATE**, inserția de înregistrări, operațiile de actualizare, indecși pentru **$PUT** și **$UPDATE**, rutina de alocare a unui grup („cluster") de blocuri și operația indexată **$CREATE.**

Segmentele de bibliotecă nu trebuie să fie fizic rezidente ; odată instalate, încărcarea acestora în memorie va avea loc dinamic, odată cu apelarea funcțiilor conținute în segmente. Pentru aplicațiile **RMS** ce utilizează numai operații asupra unor fișiere secvențiale, nu este necesară încărcarea vreunui segment satelit. După încărcarea unui segment satelit (asociat operațiilor relative la fișiere relative sau indexate), întregul suport de acces **RMS** este rezident în memorie.

Segmentele noii biblioteci de acces RMS se găsesc pe discul sistem **(LB:)** în catalogul **[1, 1]**.

14

Concepte și facilități ale rețelei MININET/MIX

14.1. Introducere în rețeaua de minicalculatoare MININET/MIX

Rețeaua **MININET/MIX** oferă posibilitatea comunicării între task-uri, partajării de resurse la distanță, controlului de task-uri la distanță, încărcării/evacuării task-urilor de la/la distanță, încărcării sistemelor la/de la distanță, comunicării între terminale, controlului și gestiunii de rețea precum și metode de acces unificate la configurația de terminale atașată unui nod, prin intermediul protocoalelor **MININET/MIX**.

MININET/MIX comunică cu nodurile adiacente prin linii de comunicație asincrone și sincrone și prin interfețe paralele.

Ca produs distribuit de rețea, **MININET/MIX** permite unui sistem de operare **MIX** (configurat în mod corespunzător) să participe, ca nod cu posibilități de rutare (nod de comutare) sau fără (nod final), în rețele de minicalculatoare formate din noduri **MIX-RT/MIX/MIX-PLUS** sau **DECNET 11M/S V3.1/V4.2, DECNET-11M PLUS V1.1/V2.2** și **DECNET/VAX.**

Accesul la rețeaua **MININET/MIX** este posibil pentru programele utilizator, scrise în limbajele de programare : **MACRO, FORTRAN IV/FORTRAN IV-PLUS, FORTRAN-77, BASIC-PLUS-2, COBOL** și **PASCAL** (vezi și cap. 40, vol. 2).

Pentru o configurare ușoară a nodurilor de rețea, **MININET/MIX** este concepută ca o structură pe mai multe nivele de protocoale, fiecare avînd un set propriu de convenții privind formatul, secvența și controlul schimbului de mesaje între protocoale.

14.2. Concepte privind topologia și tipurile structurale ale rețelei de minicalculatoare MININET/MIX

Rețeaua de minicalculatoare **MININET/MIX** este alcătuită din punct de vedere topologic din două tipuri de elemente : *noduri* și *linii de comunicație.*

Un nod de rețea **MININET/MIX** este constituit din resursele de calcul ale minicalculatoarelor din familia **INDEPENDENT** și/sau **CORAL**.

Linia de comunicație este suportul fizic de legătură între două noduri de rețea.

Dispozitivele de comunicație și *modem*-urile aferente liniilor de comunicație pot fi considerate, topologic, ca părți ale nodurilor rețelei, acestea intrînd în configurațiile sistemelor de calcul care le formează.

În rețeaua **MININET/MIX** toate nodurile pot efectua prelucrări de date, însă în funcție de locul acestora în structura rețelei pot exista următoarele tipuri de noduri : *centrale (HOST)*, *intermediare (noduri de rutare)* și *finale (satelit)*.

Nodurile centrale apar în structurile ierarhizate de rețea și cuprind în general configurații mari de calcul ce trebuie să suporte nivelele inferioare care fac apel la resursele acestuia, sau în structuri multipunct centralizat.

Nodurile satelit (finale) sînt noduri ce dispun de funcții utilizator și apar în structuri ierarhizate la nivelurile inferioare ale acestora.

Nodurile de rutare asigură rutarea adaptivă a mesajelor într-o rețea distribuită geografic.

În structuri de rețea bi-punct, oricare din cele două noduri poate deveni central și respectiv satelit, dacă dispun de resurse suficiente.

Din punct de vedere *topologic* pot exista următoarele tipuri de rețele **MININET/MIX** : *bipunct*, *multipunct*, *multipunct centralizat*, *arborescentă*, *stelară*, *inelară* și *complexă*.

Clasificarea topologică acoperă, de fapt, într-o altă formă toate tipurile funcționale descrise mai sus și evidențiază necesitățile arhitecturale ale rețelei **MININET/MIX** din fiecare nod. În raport cu poziția pe care un nod o ocupă în topologia rețelei, se poate accentua pe nivele funcționale corespunzătoare.

În principiu, nodurile centrale conțin toate cele 5 nivele funcționale ale rețelei **MININET/MIX**, deoarece asigură funcții complexe în raport cu celelalte noduri din rețea : nodurile intermediare (de intercepție sau rutare) asigură în general funcții de comutație, accentuîndu-se asupra nivelului de transport (**MITRA**) cu sau fără gestiune locală de terminale (**MITAM**) ; în nodurile satelit (finale) nivelul de transport poate dispare, în favoarea serviciilor la distanță, cu sau fără gestiune locală de echipamente periferice.

14.3. Arhitectura unui nod de rețea MININET/MIX

În rețeaua **MININET/MIX** fiecare nod de rețea (central sau satelit) are o structură multinivel, funcțiile de comunicație în rețea fiind distribuite între 7 *nivele funcționale* :

Nivelul utilizator

Cuprinde programele utilizator și serviciile de acces la rețea. Programele utilizator comunică între ele conform regulilor stabilite de utilizatori.

La acest nivel, se generează mesajele ce urmează a fi transmise între utilizatorii finali, ce pot fi : programe (task-uri), terminale, dispozitive de intrare/ieşire.

În cazul în care se doreşte comunicare între task-uri (procese), se pot utiliza, la acest nivel, directivele şi rutinele de acces la reţea standard.

Nivelul de gestiune a reţelei.

Defineşte funcţiile utilizate de operator(i) sau programe (sistem sau utilizator) pentru controlul şi menţinerea funcţionării reţelei **MININET/MIX.**

Acest nivel utilizează protocolul **NICE** (*Network Information and Control Exchange).*

Nivelul aplicaţiilor de reţea.

Defineşte funcţiile generale ale reţelei **MININET/MIX,** cum ar fi : accesul fişierelor la distanţă, transferul fişierelor la distanţă, accesul terminalelor la distanţă, etc.

Acest nivel utilizează protocolul **DAP** *Data Acces Protocol*).

În cazul în care se doreşte acces la un fişier sau la un dispozitiv plasat la distanţă, task-urile utilizator folosesc, la acest nivel, protocolul DAP (protocol de acces date).

Nivelul de control al conexiunilor logice şi serviciilor de reţea.

Defineşte un mecanism pentru crearea şi menţinerea conexiunilor logice între nivelele superioare acestuia în cadrul aceluiaşi nod sau între noduri diferite.

Task-urile utilizator şi utilitarele de reţea au acces la reţea prin intermediul conexiunilor logice ce permit o partajare centralizată a liniilor de comunicaţie între noduri. La acest nivel se asigură servicii transparente utilizatorilor, cum ar fi : pachetizare/depachetizare mesaje, alocare bufere, alocare/serializare acces conexiuni logice, precum şi funcţii generale de supraveghere a reţelei.

Acest nivel utilizează protocolul **NSP** *(Protocol Network Services*).

Nivelul de transport (rutare mesaje).

Defineşte un mecanism ce asigură transportul unui segment de date aparţinînd unui mesaj, între un nod sursă şi un nod destinaţie.

Transportul mesajelor utilizează tehnica transmiterii de pachete pe mai multe rute simultan, conform unor algoritmi de rutare adaptivi.

Acest nivel utilizează protocolul **MITRA** (**MIX** *Transport*).

Nivelul de control al conexiunilor logice şi nivelul de transport sînt utilizate deseori sub o singură denumire : *Nivelul serviciilor de reţea* (**NSP**).

Nivelul de control al legăturilor fizice.

Defineşte un mecanism pentru asigurarea unei comunicaţii fără erori între două noduri adiacente, indiferent de caracteristicile dispozitivelor şi liniilor de comunicaţie.

Acest nivel asigură transmisia efectivă a mesajelor pe liniile de comunicaţie (legături fizice), asigurînd corectarea erorilor, retransmisia de mesaje, secvenţarea mesajelor transmisie/primite, etc.

La acest nivel este implementat unul sau mai multe *protocoale de acces* pe linia de comunicaţie :

— protocoale orientate pe contor de caractere (MDLC-compatibil DDCMP) ;

— protocoale orientate caracter TMM-UC, BSC ;

— protocoale orientate bit (HDLC, SDLC, etc.).

În prima etapă de realizare a versiunii **MININET/MIX** sînt implementate protocoalele MDLC, TMM-UC și HDLC.

Nivelul fizic (hardware)

Acest nivel asigură transmisia/recepția datelor (mesajelor) pe liniile de comunicații. El este format atît din software (drivere), cît și din hardware (cuploare de transmisie, modemuri, linii de comunicație).

În paralel cu structura multinivel a rețelei **MININET/MIX** poate exista un nivel suplimentar, plasat pe poziția primelor cinci niveluri :

Sistemul de gestiune a transmisiilor

Nivelul asigură gestiunea locală sau la distanță a terminalelor atașate nodului local și o serie de funcții de serviciu pentru Nivelul de gestiune a rețelei.

Acest nivel utilizează protocolul **MITAM** (**MIX** *Teletransmission Access Method*).

Arhitectura generală a nodului de rețea **MININET/MIX** este prezentată în fig. 14.1.

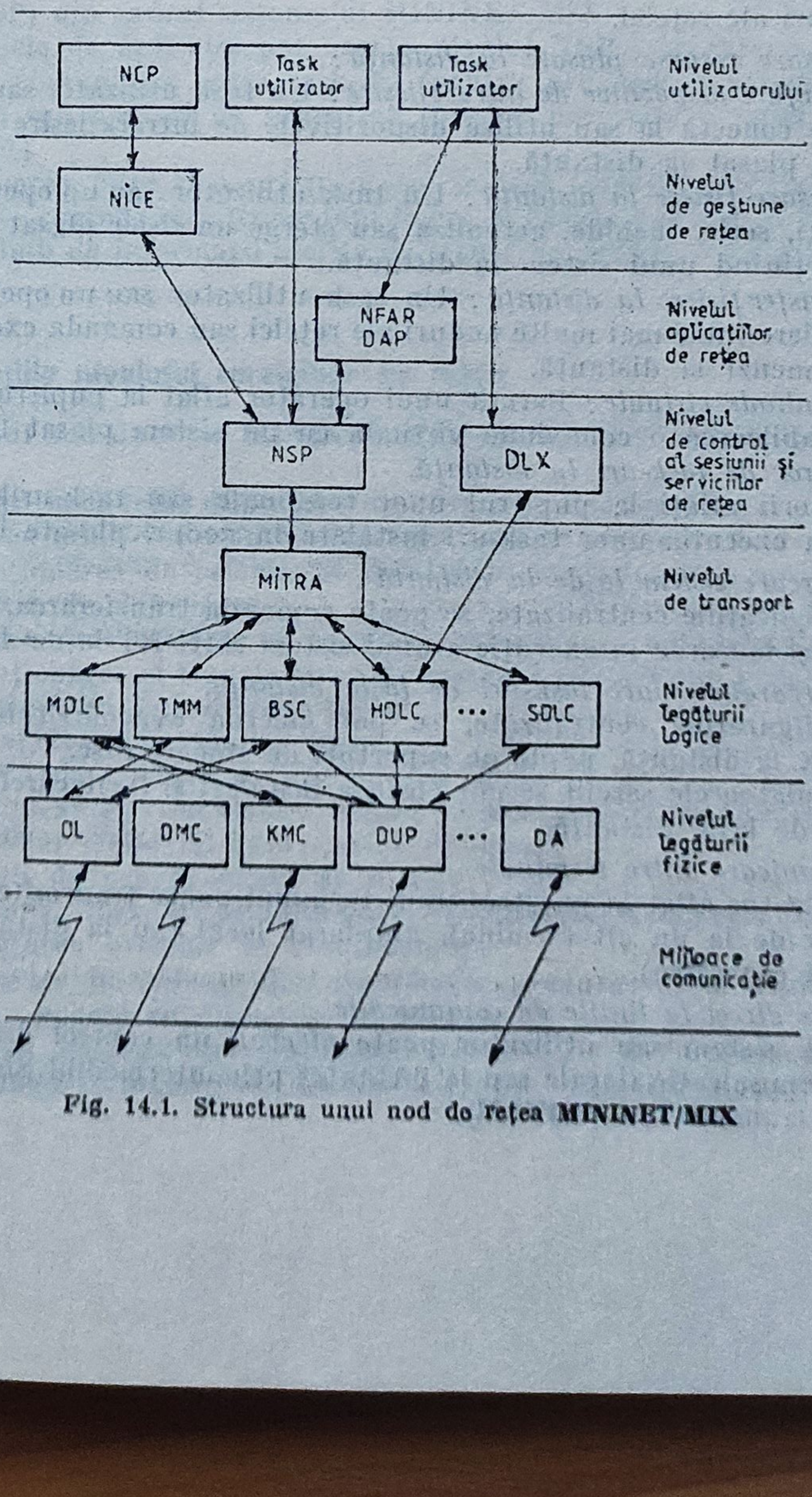


Fig. 14.1. Structura unui nod de rețea MININET/MIX

În cadrul unui nod de rețea **MININET/MIX** pot exista (împreună sau separat) niveluri conceptuale de rețea cu cel de gestiune a transmisiilor pentru efectuarea funcțiilor specifice atașate unui nod.

14.4. Posibilitățile funcționale ale rețelei de minicalculatoare MININET/MIX

Posibilitățile oferite utilizatorilor finali (programe, terminale, dispozitive) în rețeaua **MININET/MIX** sînt puse în evidență de o serie de funcții, apelabile de către utilizatori în mod direct sau indirect.

Acestea sînt :

a) *Comunicare între task-uri (procese)*

Task-urile utilizator pot comunica (prin apeluri Monitor sau de tip **FORTRAN**) date/mesaje cu alte task-uri (procese) plasate în nodul local sau în alte noduri ale rețelei.

b) *Accesare resurse plasate la distanță :*

— *Partajare dispozitive de intrare/ieșire :* Un task utilizator sau un operator se poate conecta la sau utiliza dispozitivele de intrare/ieșire aparținînd unui sistem plasat la distanță.

— *Accesare fișiere la distanță :* Un task utilizator sau un operator poate deschide, citi, scrie, închide, actualiza sau șterge un fișier plasat pe un dispozitiv aparținînd unui sistem la distanță.

— *Transfer fișiere la distanță :* Un task utilizator sau un operator poate transfera fișiere între mai multe noduri ale rețelei sau comanda execuția unui fișier de comenzi la distanță.

— *Terminale virtuale :* Permit unui operator aflat la pupitrul unui terminal să stabilească o conexiune virtuală cu un sistem plasat la distanță.

c) *Control de task-uri la distanță.*

Utilizatorii aflați la pupitrul unor terminale sau task-urile utilizator pot controla execuția unor task-uri instalate în noduri plasate la distanță.

d) *Încărcare sistem la/de la distanță*

În configurațiile centralizate, se poate comanda transferarea, încărcarea în memorie și lansarea în execuție a unui sistem MIX-RT la/de la distanță.

e) *Încărcare/evacuare task-uri de la/la distanță.*

În configurațiile centralizate, se pot efectua evacuări/reîncărcări de task-uri **MIX** la distanță, pe/de pe suporturi de stocare disc.

Pe calculatoarele satelit se pot efectua instalări și (re)încărcări/evacuări de task-uri de la/la distanță.

f) *Comunicare între terminale*

Un utilizator aflat la pupitrul unui terminal poate transmite/recepționa mesaje către/de la un alt terminal, amplasat local sau la distanță (atașat oricărui nod din rețea).

g) *Acces direct la liniile de comunicație*

Un task sistem sau utilizator poate efectua un control direct asupra liniilor de comunicație (locale sau la distanță) prin intermediul Sistemului de gestiune a transmisiilor (**MITAM**).

h) *Gestiune de rețea*

O serie de task-uri sistem gestionează și controlează componentele de rețea, testează funcționarea rețelei și raportează informații statistice și de eroare privind funcționarea nodurilor locale sau plasate la distanță.

Toate funcțiile pot fi efectuate între oricare 2 noduri ale rețelei **MININET/MIX**. Cu excepția funcțiilor de încărcare/evacuare task-uri/sisteme de la/la distanță, restul funcțiilor se pot efectua și pe configurațiile locale (pentru fiecare nod).

14.5. Concepte privind comunicarea între task-uri

Una din caracteristicile importante ale rețelei **MININET/MIX** o constituie comunicarea între task-uri (procese), plasate local (în același nod) sau la distanță (în noduri adiacente sau neadiacente).

Schimbul de mesaje între task-urile (procesele) ce comunică între ele are loc prin intermediul unor conexiuni logice, gestionate de nivelul serviciilor de rețea (**NSP**) din cadrul sistemului **MININET/MIX**.

Programele de aplicație pot accesa nivelul **NSP** prin intermediul unor directive de comunicare între task-uri (pentru programele scrise în limbajul MACRO) sau apeluri de subrutine (pentru programele scrise în limbaje de nivel înalt).

Comunicarea între task-uri (procese) se face prin intermediul unui mediu virtual de comunicație (conexiuni logice și căsuțe poștale) și a unor mijloace de schimb de informații oferite de structura multinivel a rețelei **MININET/MIX**.

14.5.1. Funcțiile nivelului serviciilor de rețea

În cadrul unui nod de rețea **MININET/MIX** nivelul **NSP** asigură următoarele funcții :

a) Funcții la nivelul dialogului cu programele utilizator :
- crearea interfeței de acces la rețea (căsuțe poștale) ;
- recepționarea de informații primite în căsuța poștală ;
- servicii de informare ;
- crearea unei interfețe de conversație între task-uri (conexiuni logice)
- recepționarea și transmiterea de mesaje pe conexiuni logice ;
- întreruperea conexiunilor logice ;
- distrugerea conexiunilor logice.

b) Funcții de operare a conexiunilor logice :
- multiplexarea conexiunilor logice pe legăturile fizice existente ;
- controlul traficului prin conexiuni logice.

c) Funcții de supraveghere de rețea :
- gestiunea fluxului de informații prin conexiuni logice ;
- detectarea erorilor în comunicație ;
- comutare în rețeaua de comunicație (comunicare cu nodurile adiacente, neadiacente sau cu ele însăși).

d) Funcții de întreținere de rețea :
- înregistrare de erori în rețea.

În continuare, se prezintă pe larg numai funcţiile şi interfeţele nivelului **NSP** cu programele de aplicaţie (aşa numita interfaţă de dialog sau de conversaţie), celelalte funcţii fiind transparente task-urilor (proceselor) ce comunică între ele.

14.5.2. Interfaţa cu nivelul aplicaţie

Pentru a avea acces la nivelul **NSP**, programele de aplicaţie solicită serviciile dorite prin intermediul unor directive de comunicaţie adresate unui dispozitiv logic de reţea, cu numele simbolic **NS:**

Serviciile furnizate se împart în 2 mari clase :

— servicii de acces la reţea ;

— servicii de comunicaţie în reţea.

Pentru accesul la reţea, nivelul **NSP** pune la dispoziţia programului de aplicaţie o cale unică prin care se pot obţine informaţii din reţea (asupra unor evenimente din reţeaua de comunicaţie care au legătură cu programul de aplicaţie respectiv).

Această cale unică, numită „căsuţă poştală“ este identificată printr-un *număr logic de acces la reţea (lunr)*, asociat anterior dispozitivului **NS:** şi precizat prin directiva **OPN$.**

Odată stabilit accesul la reţea pentru programul de aplicaţie, nivelul **NSP** poate furniza şi serviciile de comunicaţie, în cadrul reţelei, cu alt program aplicaţie. Pentru comunicare se pun la dispoziţia aplicaţiei una sau mai multe căi de comunicare (nu neapărat unice), prin care se pot transmite/recepţiona informaţii. Aceste căi numite „conexiuni logice“ sînt identificate prin *numere logice de comunicare (luni)*, asociate anterior dispozitivului **NS:** şi precizate prin directivele **CON$** sau **ACC$.**

14.5.3. Servicii de acces la reţea

Serviciile de acces ale unui program aplicaţie la reţea sînt :

— accesul aplicaţiei la reţea, crearea căsuţei poştale asociate (**OPN$**) ;

— obţinerea unor informaţii din reţea recepţionate în căsuţa poştală (**GND$**) ;

— specificarea unui mecanism asincron de atenţionare la recepţia unor informaţii din reţea în căsuţa poştală (**SPA$**) ;

— refuzarea unei cereri de creare conexiuni logică (**REJ$**), informaţie ce se va transmite pe adresa căsuţei poştale care a solicitat conectarea ;

— eliminarea accesului aplicaţiei la reţea, distrugerea căsuţei poştale asociate (**CLS$**) ;

— informarea aplicaţiei despre starea nodului curent (**GLN$**).

14.5.4. Accesul aplicaţiei la reţea

Pentru a avea acces la serviciile reţelei **MININET/MIX**, prima directivă lansată de un program de aplicaţie va fi **OPN$**. Ea va fi lansată o singură dată pe durata sesiunii de comunicare între task-uri (numim „*sesiune*“ toate schimburile de comunicaţie cuprinse între un apel **OPN$** şi altul **CLS$**).

La acceptarea directivei, nivelul serviciilor de rețea **NSP** permite accesul ulterior al aplicației la serviciile de rețea și creează căsuța poștală asociată.

În căsuța poștală vor fi recepționate toate mesajele nesolicitate adresate task-ului aplicație după acceptarea accesului acestuia la rețea. Un „*mesaj nesolicitat*" este orice mesaj recepționat de task-ul aplicație, ce nu constituie un mesaj normal de date (transmis prin **SND$** sau recepționat prin **REC$**)

Mesajele nesolicitate adresate unui task aplicație sînt plasate automat, de către **NSP**, în cutia poștală asociată.

Aceste mesaje conțin informații privind :

— cerere de stabilire conexiune logică — din partea altui task aplicație pentru conectare **(CON$)** ;

— mesaje de întrerupere — transmisie în afara și în paralel cu fluxul normal de mesaje **(XMI$)** ;

— deconectare sau terminare anormală a unei conexiuni logice — din partea unui alt task aplicație, ce atrage atenția asupra stării unei conexiuni logice : a fost deconectată **(DSC$)** sau terminată anormal **(ABT$)** ;

— deconectare a unei conexiuni logice — din partea nivelului serviciilor de rețea **(NSP)**.

În cazul în care s-a specificat un mecanism de atenționare la recepția mesajelor nesolicitate, la fiecare nouă introducere a unui mesaj în cutia poștală se pasează controlul unei rutine de tratare **AST** specificată de directiva **SPA$**.

Indiferent de faptul că s-a specificat sau nu un mecanism de atenționare, mesajele nesolicitate pot fi extrase din cutia poștală prin programarea unei directive **GND$**.

14.5.5. Extragerea mesajelor nesolicitate

Mesajele nesolicitate primite de către un task aplicație sînt plasate, de către **NSP**, în cutia poștală asociată, în ordine FIFO (primul venit, primul servit). La extragerea unui mesaj din cutia poștală (**GND$**), acesta va fi plasat într-un buffer specificat de utilizator și eliminat din cutie.

Există *două moduri* prin care pot fi selectate mesajele nesolicitate :

a) *în ordinea sosirii* (FIFO). În apelul de directivă se specifică numai adresa și lungimea buferului de recepție, în care va fi introdus primul mesaj din coadă ;

b) *prin selecție*, după anumite informații de apel. În apelul de directivă se specifică adresa și lungimea bufer-ului de recepție, tipul de mesaj și numărul conexiunii logice pe care s-a recepționat mesajul.

Primul mesaj, la care corespund tipul și numărul de conexiune logică specificate, va fi extras și introdus în bufer-ul specificat. Întrucît un task poate comunica simultan cu alte task-uri pe mai multe conexiuni logice, toate mesajele nesolicitate parvenite de la aceste conexiuni sînt plasate în cutia poștală unică.

Această opțiune permite selectarea precisă a mesajului nu numai după tip (mesaj întrerupere, mesaj cerere conectare, deconectare sau terminare anormală utilizator, deconectarea **NSP**), ci și după conexiunea logică pe care

s-a recepționat mesajul. În cazul în care se omite numărul logic de conexiune se extrage primul mesaj identic cu tipul specificat (indiferent de conexiunea logică pe care a fost recepționat).

14.5.6. Eliminarea accesului aplicației la rețea

La programarea unei directive **CLS$**, se termină sesiunea de comunicație între task-urile aplicație și se elimină accesul task-ului apelant de la serviciile de rețea.

Numărul logic specificat în apelul **CLS$** este cel precizat la deschiderea sesiunii (**OPN$**). La închiderea sesiunii nivelul **NSP** execută următoarele acțiuni :

— forțează terminarea anormală a tuturor conexiunilor logice stabilite de către task-ul aplicație ;

— eliberează toate numerele logice asociate acestor conexiuni ;

— elimină toate mesajele de întrerupere și de deconectare prezente în căsuța poștală ;

— distruge căsuța poștală asociată task-ului.

Reluarea sesiunii de comunicație e posibilă numai prin apelarea unei noi directive **OPN$**.

14.5.7. Servicii de comunicație în rețea

Serviciile de comunicație în rețea ale unui program aplicație sînt :

— solicitare creare conexiune logică (**CON$**) între task-ul apelant (sursă) și un alt task din rețea (destinație) ;

— acceptare cerere de creare conexiune logică (**ACC$**) ;

— transmisie date pe conexiune logică (**SND$**) ;

— recepție date pe conexiune logică (**REC$**) ;

— transmitere mesaj de întrerupere pe conexiune (**XMI$**) ;

— deconectare (distrugere) conexiune logică, după rezolvarea tuturor cererilor de comunicare pe conexiune (**DSC$**) ;

— distrugere conexiune logică, indiferent de starea cererilor de comunicare pe conexiune (**ABT$**).

14.5.8. Conexiuni logice

După stabilirea unei sesiuni de comunicație (permitere acces la rețea — **OPN$**), task-ul aplicație poate începe o serie de „conversații" cu alte noduri ale rețelei **MININET/MIX**.

Conversația (comunicarea) are loc prin intermediul unor *conexiuni logice*, ce constituie căi discrete de conversație pentru o pereche de task-uri. Odată stabilită conexiunea logică, cele două task-uri pot trimite și recepționa mesaje specificînd numai numărul logic atașat acestei conexiuni, eliminîndu-se identificarea prin nume, acces, etc.

Acest *număr logic* este asociat conexiunii logice la crearea acesteia, fiind diferit de cel de acces la rețea.

Trebuie accentuat asupra faptului că fiecare task ce solicită comunicare în rețea trebuie să specifice minimum două numere logice :

— unul pentru accesul la rețea ;

— altul pentru fiecare conexiune logică activă.

Fiecare conexiune logică poate opera în mod duplex integral, putîndu-se recepționa și emite mesaje în paralel.

Rețeaua **MININET/MIX** permite de asemenea unui task aplicație să comunice cu mai multe alte task-uri, simultan, prin conexiuni logice multiple sau să creeze conexiuni multiple, separate, cu un singur task.

Pentru a înțelege modul în care se creează o conexiune logică, task-urile cooperante (ce urmează a conversa între ele) sînt referite sub numele de „sursă" și „destinație". Task-ul care solicită conectarea (emite o cerere de stabilire conexiune — **CON$**) se numește „task sursă", iar task-ul care acceptă (**ACC$**) sau refuză (**REJ$**) cererea de conectare se numește „task destinație".

Distincția între sursă și destinație este temporară și se referă numai la procesul de conectare (de stabilire a conexiunii logice). Odată stabilită conexiunea, cele două task-uri au un statut egal, fiecare putînd emite/recepționa informații sau deconecta/termina anormal conexiunea.

Pentru *crearea unei conexiuni logice* cu task-ul destinație, task-ul sursă execută directiva **CON$**. Pentru fiecare nouă conexiune logică se execută **CON$**, specificînd un număr logic de conexiune diferit. Acest număr logic, specificat la **CON$**, va constitui informația de identificare a conexiunii logice și va fi utilizat în toate serviciile ulterioare de comunicare (**SND$**, **REC$**, **XMI$**, **DSC$** sau **ABT$**).

Pentru stabilirea unei conexiuni, task-ul sursă va furniza o serie de informații utilizate la identificarea task-ului aplicație destinatar. Aceste informații, stocate într-un bloc de cerere conexiune (CRB), descris de macroinstrucțiunea **CONB$$**, vor fi utilizate de către cele două nivele ale serviciilor de rețea (**NSP** sursă și **NSP** destinație) pentru crearea și operarea unei conexiuni logice.

Informațiile din blocul de cerere conexiune identifică task-ul destinatar în :

— rețeaua **MININET/MIX** — prin precizarea nodului în care este plasată aplicația ;

— nodul de rețea localizat — prin precizarea numelui de task și al drepturilor de acces la acesta.

Adresa blocului de cerere conexiune este plasată în apelul directivei **CON$**.

Cererea și blocul de stabilire conexiune sînt transmise de către nivelul **NSP** din nodul sursă, către nivelul **NSP** din nodul destinație.

Acesta, verifică mai întîi informațiile de identificare ale task-ului sursă (conținute în blocul de cerere conectare) pentru a determina dacă task-ul sursă are drept de acces la acest nod.

Dacă nu are drepturi de acces sau task-ul destinatar nu este instalat, *se refuză conectarea* (**REJ$**).

Dacă accesul este permis și task-ul destinatar specificat este activ, nivelul **NSP** destinație plasează cererea de stabilire conexiune în căsuța poștală atașată acestui task.

Dacă task-ul destinatar este instalat, dar inactiv, nivelul **NSP** destinație cere lansarea în execuție a acestuia. După lansarea în execuție, task-ul destinatar va solicita acces la rețea prin **OPN$**. La această solicitare, nivelul **NSP** destinație creează cutia poștală asociată task-ului destinație și inserează în ea cererea de stabilire conexiune.

Această cerere este extrasă de task-ul destinație prin apelarea directivei **GND$**, specificînd același număr logic cu **OPN$**.

După preluarea pachetului conținînd cererea de stabilire conexiune, task-ul destinatar poate :

— *accepta cererea de stabilire conexiune* (**ACC$**). Se creează o conexiune logică între cele două task-uri. Task-ul sursă este atenționat, prin blocul de stare al **CON$**, de acceptarea cererii de stabilire conexiune de către task-ul destinatar. Din acest moment, atît task-ul sursă cît și cel destinatar pot solicita servicii de comunicație în rețea (**SND$**, **REC$**, **XMI$**, **DSC$**, **ABT$**), specificînd drept număr logic al conexiunii numărul precizat în cererea de conectare (**CON$**).

— *refuză cererea de stabilire conexiune* (**REJ$**). În cazul în care task-ul destinatar refuză cererea de stabilire conexiune, în apelul directivei **REJ$** se utilizează numărul logic atașat căsuței poștale a task-ului destinatar. Task-ul sursă conexiune este atenționat de refuzarea cererii de stabilire conexiune prin intermediul blocului de stare al **CON$**.

14.5.9. Transmiterea/recepția de mesaje pe conexiuni

După stabilirea unei conexiuni logice, cele două task-uri (sursă și destinație) devin task-uri cooperante, amîndouă putînd controla conexiunea logică prin transmiterea (**SND$**)/recepția (**REC$**) de mesaje de date, transmiterea unor mesaje nesolicitate sau pot deconecta/termina anormal comunicația pe conexiune.

14.5.10. Transmiterea de mesaje de întrerupere

Oricînd, după stabilirea unei conexiuni, unul din cele două task-uri cooperante poate transmite un mesaj de întrerupere către celălalt (**XMI$**).

Lungimea mesajelor de întrerupere este limitată la 16 caractere, aceste mesaje fiind utilizate pentru informarea celuilalt task (semnalizare) despre apariția unor condiții neașteptate în execuția task-ului curent. Nivelul **NSP** asociat nodului în care se găsește task-ul destinatar plasează mesajul de întrerupere în cutia poștală asociată acestui task.

Pentru extragerea mesajului de întrerupere, task-ul destinatar trebuie să emită o directivă **GND$**.

14.5.11. Deconectarea/terminarea unei conexiuni logice

Oricînd, după stabilirea unei conexiuni logice, unul din cele două task-uri cooperante poate solicita deconectarea/terminarea anormală a unei conexiuni logice.

În acest caz, se poate utiliza :

— directiva **DSC$** — ce solicită deconectarea task-ului de la o conexiune logică. Aceasta reprezintă o cale normală de terminare a comunicaţiei pe o conexiune logică. Toate cererile de transmisie în curs sau în aşteptare vor fi completate, înainte de începerea propriu-zisă a operaţiei de deconectare.

— directiva **ABT$** — ce solicită deconectarea task-ului de la o conexiune logică indiferent de starea transmisiilor în curs. Toate cererile de emisie şi recepţie vor fi terminate anormal înainte de începerea operaţiei de deconectare.

În cazul în care deconectarea/terminarea anormală este solicitată de task-ul sursă, în apelul acestor directive se va include numărul logic specificat în cererea de stabilire a conexiunii (**CON$**).

În cazul în care deconectarea/terminarea anormală este solicitată de task-ul destinaţie, în apelul acestor directive se va include numărul logic specificat la acceptarea cererii de stabilire a conexiunii (**ACC$**).

După deconectarea/terminarea anormală a unei conexiuni logice, se eliberează numărul logic asociat acesteia, putînd fi utilizat ulterior într-o nouă operaţie de conectare (**CON$**).

În figura 14.2 este prezentată schematic, modalitatea concretă de creare a unei conexiuni logice între două task-uri din reţea.

14.6. Componentele reţelei MININET/MIX

Funcţiile reţelei de minicalculatoare MININET/MIX sînt asigurate de următoarele *4 categorii de programe :*

Programe de bază şi control ale reţelei de minicalculatoare

Programele de bază şi control sînt implementate în următoarele moduri :

a) ca *procese sistem,* următoarele programe :

— Monitor de comunicaţii (**CEX, DDHAR, STCRC**) ;

— Monitor control modemuri (**AUX**) ;

— Monitor servicii de reţea (**NSP**) ;

— Sistem de gestiune a transmisiilor de reţea (**DLX**) ;

— Protocoale de comunicaţii pe linie (**DLC**) : **MDLC** (compatibil **DDCMP**), **HDLC** şi **MOP.**

— Drivere pentru liniile de comunicaţie (**DDM**) : **DL, DUP, DZ, DMC, KMC—DUP, KMC—DZ** şi **DA.**

— Încărcătoare primare/secundare/terţiare (**MOP**) ;

b) ca *task-uri sistem,* următoarele programe :

— Procesor ajutător pentru servicii de reţea (**NETACP**) ;

— Procesor de control reţea, cu variante :

— **NCP** (central, funcţii maxime) ;

— **NCP** (central/satelit, funcţii medii) ;

— **NCP** (satelit, funcţii minime).

— Program auxiliar de control reţea (**NMVACP. NICE**) ;

— Program de încărcare în memorie componente reţea (**NTL**) ;

— Program de încărcare disc componente reţea (**VNP**) ;

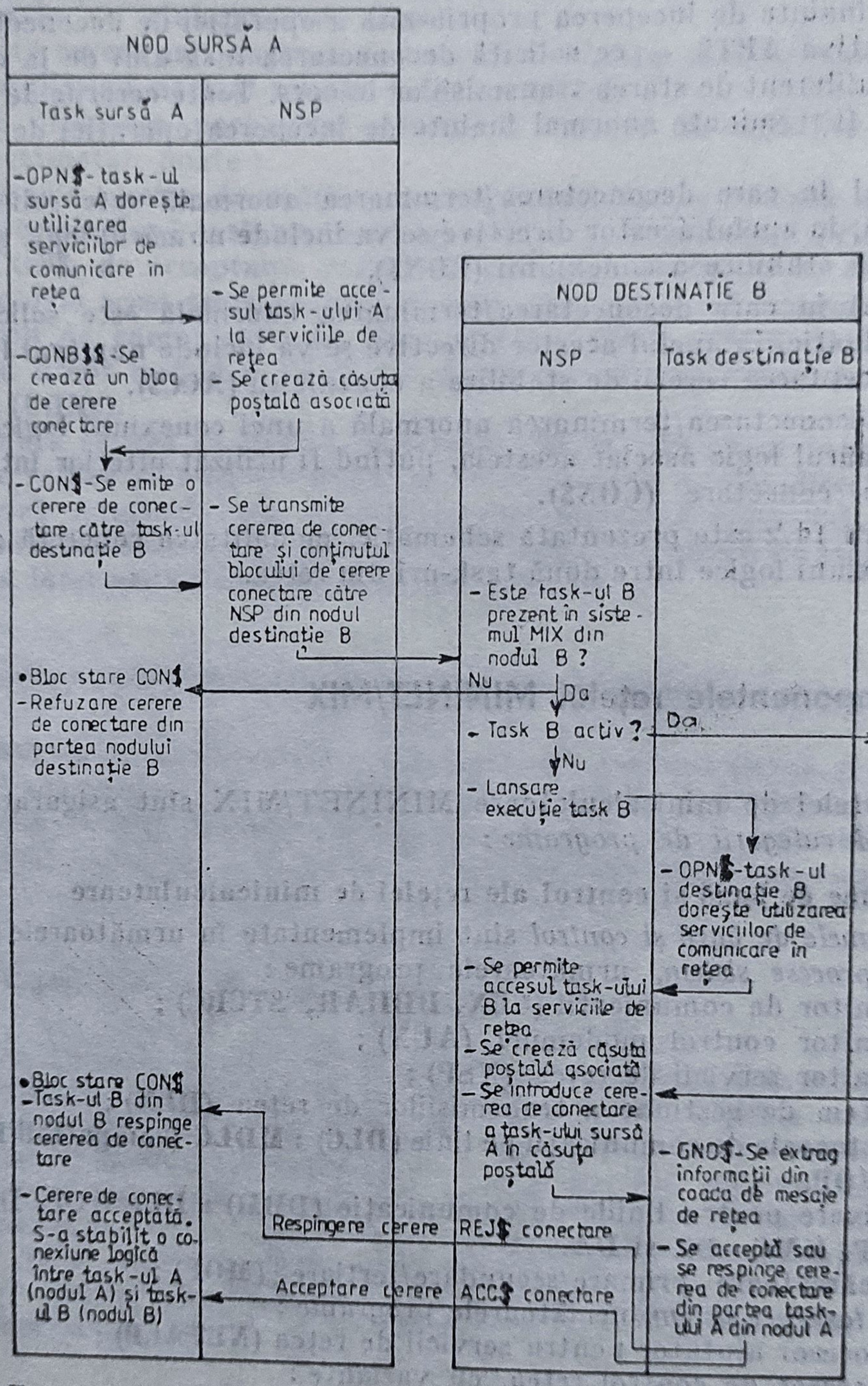


Fig. 14.2. Modalitatea de creare a unei conexiuni logice între două task-uri dintr-o rețea MININET/MIX

— Program de inițializare componente rețea (**NTINIT**) ;
— Program de încărcare/evacuare task-uri de la/la distanță :
  — **HLD** (central/satelit cu disc) ;
  — **SLD** (satelit rezident în memorie).

Programe utilitare de rețea

Programele utilitare de rețea folosesc serviciile și conceptele oferite de programele de bază ale rețelei (procese și task-uri) :

— utilitare pentru controlul execuției task-urilor la distanță (**TCL, RAC**) ;
— utilitare pentru acces și transfer de fișiere la distanță (**FAL, NFT, MCM, FTS, FTSDEQ**) ;
— utilitare pentru comunicare între terminale (**TLK, LSN**) ;
— programe de test al rețelei **MININET/MIX** (**DTS, DTR**) ;
— program de vizualizare tabele interne de descriere rețea (**CEDUMP**) ;
— program de editare structură descriere rețea (**CFE**).

Biblioteci de acces la rețea

Permit implementarea ușoară și standardizată a aplicațiilor de rețea, definind o serie din funcțiile generale rețelei **MININET/MIX** :

— rutine pentru accesul/transferul fișierelor la distanță (**DAP, DAPRMS**) ;
— rutine de acces FORTRAN privind comunicarea între task-uri și controlul task-urilor la distanță (**NETFOR.OLB**) ;
— rutine generale de acces la rețea (**NETLIB.OLB**) ;
— macroinstrucțiuni de rețea, incluzînd directivele de comunicare între task-uri și interfețe de apel **DLX** (**NETLIB.MLB**).

Generatoare de rețea

Generarea rețelei **MININET/MIX** cuprinde două faze importante :
— planificarea/configurarea unui nod din rețea (**NETPLN**) ;
— generarea (automată sau autonomă) a componentelor de rețea.

Generarea se poate face sub controlul generatoarelor de sistem **MIX/(MIX-PLUS)** sau separat (după generarea sistemului suport de rețea).

14.7. Variante ale rețelei MININET/MIX

Pentru realizarea arhitecturii de rețea **MININET/MIX**, au fost concepute variante specifice familiei de sisteme **MIX** :

— **MININET/MIX** = sistem distribuit de rețea sub **MIX V2.0**, compatibil cu pachetul de rețea **DECNET-11M V3.1** al firmei DEC, SUA ;
— **MININET/MIX-PLUS** = sistem distribuit de rețea sub **MIX-PLUS V1.0**, compatibil cu pachetul de rețea **DECNET-11M PLUS V1.1** al firmei SUA ;
— **MININET/MIX-RT** = sistem distribuit de rețea sub **MIX-RT V2.0**, compatibil cu pachetul de rețea **DECNET-11S V3.1**, al firmei DEC, SUA ;
— **MININET/DLX** = sistem minimal de asigurare a comunicațiilor intercalculatoare, compatibil cu pachetul de comunicații **DLX-11** al firmei DEC, SUA.

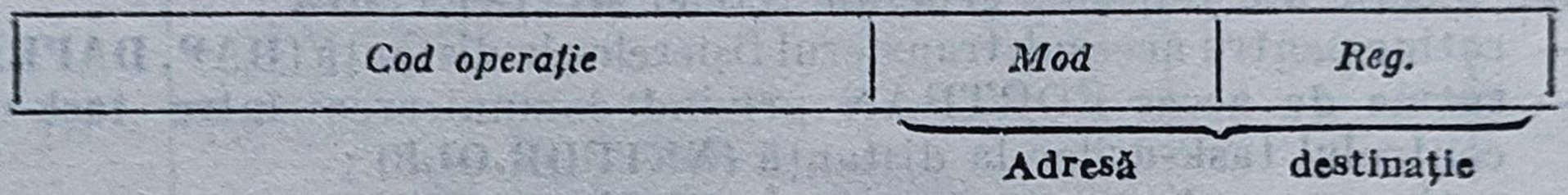
INSTRUCȚIUNILE LIMBAJULUI DE PROGRAMARE MACRO

Tabela A.1

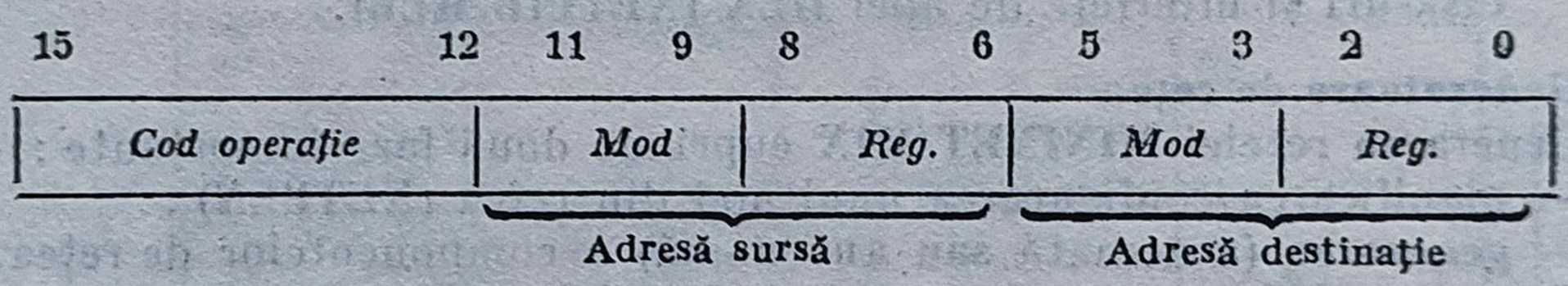
FORMATELE GENERALE ALE INSTRUCȚIUNILOR MINICALCULATOARELOR ROMÂNEȘTI

A. FORMATE INTERNE CALCULATOR

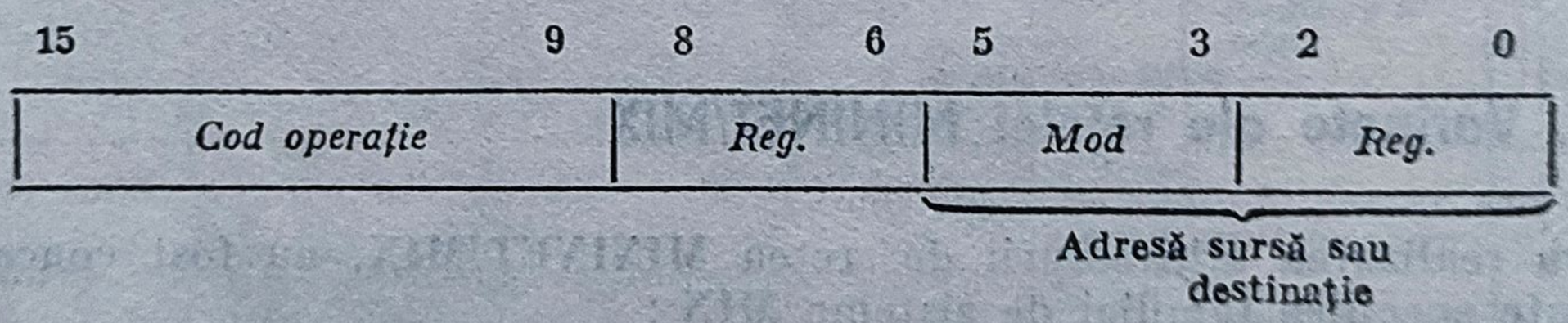
1. Instrucțiuni cu un singur operand:



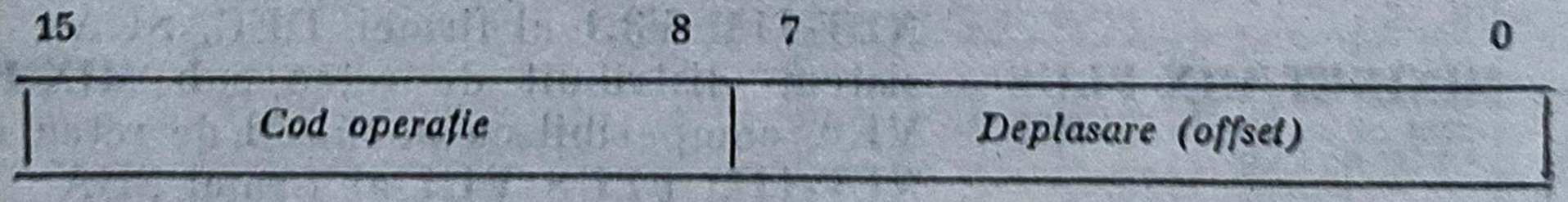
2. Instrucțiuni cu doi operanzi:



3. Instrucțiuni Registru — sursă sau destinație:

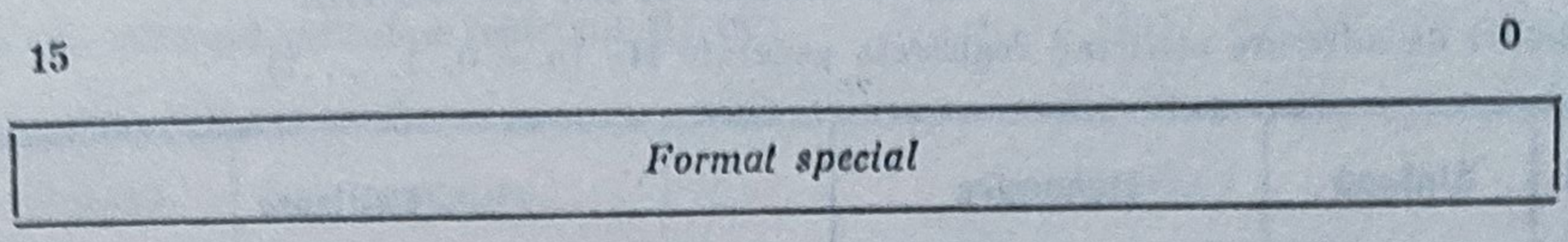


4. Instrucțiuni de salt:

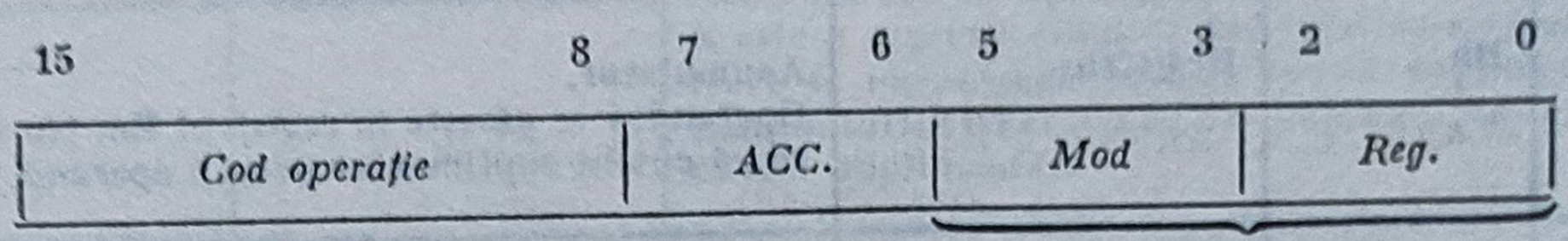


5. Instrucțiuni asupra codurilor de condiții:

6. Instrucţiuni cu format special :

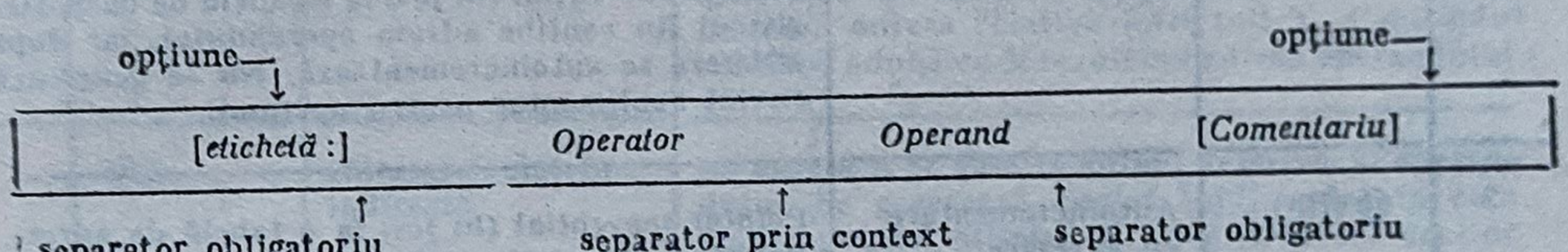


7. Instrucţiuni do virgulă mobilă lungă (FPP) :



B. FORMATE SURSĂ ALE LIMBAJULUI MACRO

1. Formatul general al instrucţiunilor **MACRO** :



2. Instrucţiuni **MACRO** cu un operand :

[Etichetă :]	Mnemonică instrucţiune	Operand*	[; Comentariu]

* Sau o expresie de calcul al deplasării (offset), în cazul instrucţiunilor de salt.

3. Instrucţiuni **MACRO** cu doi operanzi :

[Etichetă :]	Mnemonică instrucţiune	Operand sursă,	Operand destinaţie	[; Comentariu]

4. Instrucţiuni **MACRO** de prelucrări în virgulă mobilă **(FPP)**

4.1. Cu un operand :

(Etichetă :]	Mnemonică instrucţiune	Operand**	[; Comentariu]

4.2. Cu doi operanzi :

[Etichetă :]	Mnemonică instrucţiune	Acumulator,	Operand**	[; Comentariu]

** Operandul poate fi : sursă/destinaţie mod adresare **UC**
sursă/destinaţie mod adresare **FPP**

Tabela A.2

SUMAR AL MODURILOR DE ADRESARE

1. Moduri de adresare utilizînd registrele generale R*n* ($n = 0, 1, \ldots 7$)

Mod	Sintaxă	Denumire	Utilizare
1	2	3	4
0	Rn	Registru	Acumulator. Operandul se găsește în registrul R*n*. Nu se generează cuvînt suplimentar pentru operand.
1	(R*n*) sau @R*n*	Registru indirect	Pointer la o adresă. Registrul R*n* conține adresa operandului și rămîne nemodificat după adresare. Nu se generează cuvînt suplimentar pentru operand.
2	(R*n*)+	Autoincrementare	Pointer secvențial (în jos) la o tabelă de date. Registrul R*n* conține adresa operandului, iar după adresare se autoincrementează. Nu se generează cuvînt suplimentar pentru operand.
3	@(R*n*)+	Autoincrementare cu adresare indirectă	Pointer secvențial (în jos) la o tabelă de adrese. Registrul R*n* conține adresa adresei operandului și după adresare se autoincrementează. Nu se generează cuvînt suplimentar pentru operand.
4	−(R*n*)	Autodecrementare	Pointer secvențial (în sus) la o tabelă de date. Registrul R*n* se autodecrementează și apoi indică adresa operandului. Nu se generează cuvînt suplimentar pentru operand.
5	@ −(R*n*)	Autodecrementare cu adresare indirectă	Pointer secvențial (în sus) la o tabelă de adrese. Registrul R*n* se autodecrementează și apoi indică adresa operandului. Nu se generează cuvînt suplimentar pentru operand.
6	X(R*n*)	Indexat	Acces direct într-o tabelă de date. Conținutul registrului R*n* e adună cu valoarea expresiei X, formînd adresa operandului. R*n* rămîne nemodificat. Se generează cuvînt suplimentar în care se memorează valoarea expresiei X.
7	X @(R*n*)	Indexat cu adresare indirectă	Acces direct într-o tabelă de adrese. Conținutul registrului R*n* se adună cu valoarea expresiei X, formînd adresa adresei operandului. R*n* rămîne nemodificat. Se generează cuvîntul suplimentar în care se memorează valoarea expresiei X.

Tabela A.2 (continuare)

2. Moduri de adresare utilizînd registrul R7(PC)

Mod	Sintaxă	Denumire	Utilizare
2	#X	Adresare imediată	Acces direct la operandul **X**. **X** este o expresie absolută sau relativă a cărei valoare se memorează într-un cuvînt suplimentar. Conținutul lui **R7** crește cu 2 și indică adresa acestui cuvînt suplimentar.
3	@#X	Adresare absolută	Acces imediat la o adresă **X**. Valoarea expresiei **X** este memorată într-un cuvînt suplimentar și reprezintă adresa absolută a operandului. Registrul **R7** conține adresa adresei operandului.
6	X	Adresare relativă	**X** reprezintă adresa relativă a operandului. Se generează un cuvînt suplimentar în care se memorează adresa relativă (offsetul) **X**. Conținutul lui **R7** se adună cu **X** rezultînd adresa operandului.
7	@X	Adresare relativă indirectă	La adresa **X** se găsește adresa relativă a operandului. Se generează cuvînt suplimentar în care se memorează offsetul **X**. Registrul **R7** conține adresa adresei operandului.

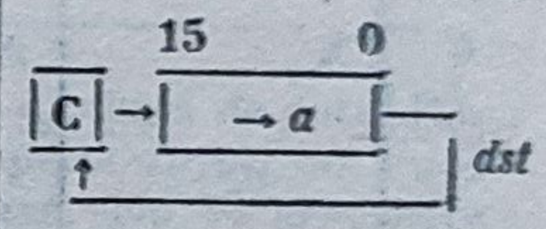
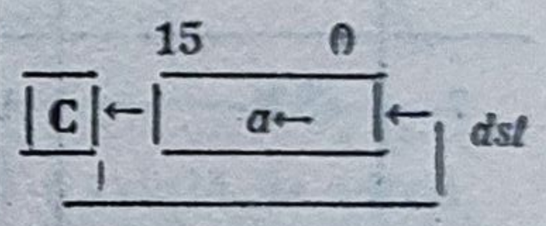
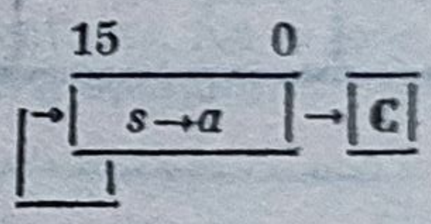
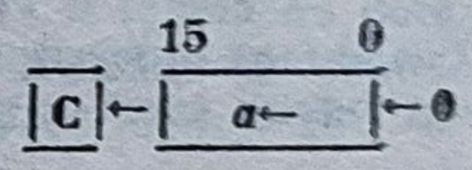
Tabela A.3

SETUL STANDARD (DE BAZĂ) DE INSTRUCȚIUNI MACRO
CONVENȚII DE NOTARE :

SS = element de adresă pentru operandul sursă
DD = idem, pentru operandul destinație
srs = conținutul operandului sursă
dst = conținutul operandului destinație
R*n* = registrul *n* ($n = 1, 2, \ldots, 7$)
(B) = instrucțiune cu operand pe octet
■ = { 0 pentru operand pe cuvînt / 1 pentru operand pe octet
ZNVC = indicatorii de condiție : Zero, Negativ, Depășire (Overflow), Transport (Carry)
* = valoare 0 sau 1 (pentru **ZNVC**)
– = indicator de condiție neafectat
→ = atribuire (mutare)
a = valoarea bitului care prin deplasare sau rotație se atribuie indicatorului de condiție **C**
∪ = funcție logică SI
∧ = funcția logică SAU INCLUSIV
⊕ = funcția logică SAU EXCLUSIV
~ = funcția logică NOT

Tabela A.3 (continuare)

1.1. *Instrucțiuni cu un singur operand*

Mnemonica	Denumire instrucțiune	Cod operație (octal)	Poziționare indicatori NZVC	Funcția
1	2	3	4	5
CLR(B)	Ștergere	050DD	0 1 0 0	0 → *dst*
COM(B)	Complementare valoare (față de unu)	051DD	* * 0 1	~ *dst* → *dst*
INC(B)	Incrementare	052DD	* * * —	*dst* + 1 → *dst*
DEC(B)	Decrementare	053DD	* * * —	*dst* − 1 → *dst*
NEG(B)	Negare (complement față de doi)	054DD	* * * *	-*dst* → *dst*
TST(B)	Testare	057DD	* * 0 0	Poziționează NZVC în funcție de semnul lui *dst* (care rămîne neschimbată)
ROR(B)	Rotație dreaptă (cu 1 bit)	060DD	* * * a	
ROL(B)	Rotație stînga (cu 1 bit)	061DD	* * * a	
ASR(B)	Deplasare aritmetică la dreapta (cu 1 bit)	062DD	* * * a	
ASL(B)	Deplasare aritmetică la stînga (cu 1 bit)	063DD	* * * a	
SWAB	Interschimbare octeți	0003DD	* * * 0	Se schimbă octeții unui cuvînt între ei (*dst* trebuie să aibă adresă pară).
ADC(B)	Adunare transport	055DD	* * * *	*dst* + C → *dst*
SBC(B)	Scădere transport	056DD	* * * *	*dst* − C → *dst*
SXT	Extensie semn cuvînt	0067DD	− * 0 −	N → *dst* (0 → *dst* dacă N = 0; −1 → *dst* dacă N=1)

1.2. *Instrucțiuni cu doi operanzi*

1	2	3	4	5
MOV(B)	Transfer	1SSDD	* * 0 —	*srs* → *dst*
CMP(B)	Comparare	2SSDD	* * * *	*srs* − *dst* ⇒ NZVC (*srs*, *dst* rămîn nemodificate)

Tabela A.3 (continuare)

1	2	3	4	5
ADD	Adunare	06SSDD	* * * *	*dst* + *srs* → *dst*
SUB	Scădere	16SSDD	* * * *	*dst* − *srs* → *dst*
BIT(B)	Testare biți	■3SSDD	* * 0 —	*dst* ∧ *srs* ⇨ **NZ** (*srs*, *dst* rămîn nemodificate)
BIC(B)	Ștergere biți (Funcție INHIBIȚIE)	■4SSDD	* * 0	$\sim \overline{srs} \cup dst \rightarrow dst$
BIS(B)	Poziționare biți Funcție SAU)	■5SSDD	* * 0 —	*dst* ∧ *srs* → *dst*
XOR	Funcția logică SAU EXCLUSIV	074RDD	* * 0 —	**R** ⊕ *dst* → *dst* *srs* trebuie să fie un registru

1.3. *Instrucțiuni de salt*

Mnemonica	Denumire instrucțiune	Cod operație (octal)	Funcția
1	2	3	4
	a) Salt condiționat care testează un indicator de condiție		
BNE	Salt dacă diferit (de zero)	001000	Salt dacă **Z** = 0 (sau dacă *srs* ≠ *dst*)
BEQ	Salt dacă egal (cu zero)	001400	Salt dacă **Z** = 1 (sau dacă *srs* = *dst*)
BPL	Salt dacă pozitiv	100000	Salt dacă **N** = 0
BMI	Salt dacă negativ	100400	Salt dacă **N** = 1
BVC	Salt dacă nu există depășire	102000	Salt dacă **V** = 0
BVS	Salt dacă există depășire	102400	Salt dacă **V** = 1
BCC	Salt dacă nu există transport	103000	Salt dacă **C** = 0 (sau dacă *srs* ≥ *dst*)
BCS	Salt dacă există transport	103400	Salt dacă **C** = 1 (sau dacă *srs* < *dst*)
	b) Salt condiționat după operații algebrice (cu semn)		
BNE, BEQ	— vezi mai sus		
BGE	Salt dacă este mai mare sau egal (cu zero)	002000	Salt dacă „≥". (**N** ⊕ **V** = 0)
BGT	Salt dacă este mai mare decît (zero)	003000	Salt dacă „>". (**Z** ∧ (**N** ⊕ **V**) = 0)
BLE	Salt dacă este mai mic sau egal (cu zero)	003400	Salt dacă „≤". (**Z** ∧ (**N** ⊕ **V**) = 1)
BLT	Salt dacă este mai mic decît (zero)	002400	Salt dacă „<". (**N** ⊕ **V** = 1)
	c) Salt condiționat după operații aritmetice (fără semn)		
BNE, BEQ	— vezi mai sus		
BCC, BCS	— vezi mai sus		
BHIS	Salt dacă este mai mare sau egal	103000	Salt dacă „>" (sau dacă *srs* ≥ *dst*; **C** = 0)

Tabela A.3 (continuare)

1	2	3	4
BHI	Salt dacă este mai mare	101000	Salt dacă „>" (sau dacă *srs* > *dst* ; **C** = 0 și **Z** = 0)
BLOS	Salt dacă este mai mic sau egal	101400	Salt dacă „>" (sau dacă *srs* ⩽ *dst* ; **C** ∧ **Z** = 1)
BLO	Salt dacă este mai mic	103400	Salt dacă „<" (sau dacă *srs* < *dst* ; **C** = 1)

d) *Salt necondiționat*

BR	Salt necondiționat	000400	(**PC** ← **PC** + 2 × offset)
JMP	Salt necondiționat; destinație definită	001DD	*dst* → **PC**
SOB	Decrementare și salt (dacă diferit zero)	077RNN	**R** − 1 → **R** ; dacă rezultatul ≠ 0, atunci **PC** ← **PC** − 2 × offset (salt)

1.4. *Instrucțiuni de salt și retur la/din subrutine*

Mnemonic	Denumire instrucțiune	Cod operație (octal)	Funcție
JSR	Salt la subrutină	004RDD	*dst* → **PC**
RTS	Revenire din subrutină	00020R	**R** → **PC**
MARK	Marcare retur din subrutină	0064NN	

1.5. *Instrucțiuni pentru tratarea derutărilor și întreruperilor*

Mnemonica	Instrucțiune	Cod operație (octal)
EMT	Apel Monitor (utilizare rezervată)	104000—104377
TRAP	Derută pe nivelul de întrerupere 4	104400—104777
BPT	Derută de trasare	000003
IOT	Derută de intrare/ieșire	000004
RTI	Retur din tratare întrerupere	000002
RTT	Retur din tratare întrerupere cu inhibare trasare	000006

1.6. *Instrucțiuni diverse*

Mnemonica	Instrucțiune	Cod operație (octal)
1	2	3
HALT	Oprire procesor	000000
WAIT	Așteptare apariție întrerupere	000001
RESET	Inițializare generală **BUS**	000005
NOP	Operație vidă	000240
SPL	Stabilire nivel prioritate (în **PS**)	00023N

Tabela A.3 (continuare)

1	2	3
MFPI	Transfer din spaţiul de instrucţiuni anterior în spaţiul curent	0065SS
MTPI	Transfer în spaţiul de instrucţiuni anterior din spaţiul curent	0066DD
MFPD	Transfer din spaţiul de date anterior în spaţiul curent	1065SS
MTPD	Transfer în spaţiul de date anterior din spaţiul curent	1066DD
MFPS	Transfer din cuvîntul de stare program (PS)	1067DD
MTPS	Transfer în cuvîntul de stare program (PS)	1064SS

unde: N — nivel de prioritate

1.7. *Instrucţiuni asupra indicatorilor de condiţii*

Mnemonica	Instrucţiune	Cod operaţie (octal)
CLC	Ştergere inidicator **C**	000241
CLV	Ştergere indicator **V**	000242
CLZ	Ştergere indicator **Z**	000244
CLN	Ştergere indicator **N**	000250
CCC	Ştergere toţi indicatorii de condiţii	000257
SEC	Setare indicator **C**	000261
SEV	Setare indicator **V**	000262
SEZ	Setare indicator **Z**	000264
SEN	Setare indicator **N**	000270
SCC	Setare toţi indicatorii de condiţii	000277

Tabela A.4

SET INSTRUCŢIUNI ARITMETICE EXTINSE (EIS)

Mnemonica	Denumire instrucţiune	Cod operaţie (octal)	Poziţionare indicatori N Z V C	Funcţia
1	2	3	4	5
MUL	Înmulţire	070RSS	* * 0 *	R, R × *srs* → R, R + 1 dacă R = par R, R × *srs* → R dacă R = impar; *dst* trebuie să fie un registru
DIV	Împărţire	071RSS	* * * *	R, R + 1/*srs* → → {R = cît {R + 1 = rest *dst* trebuie să fie un registru par

Tabela A.4 (continuare)

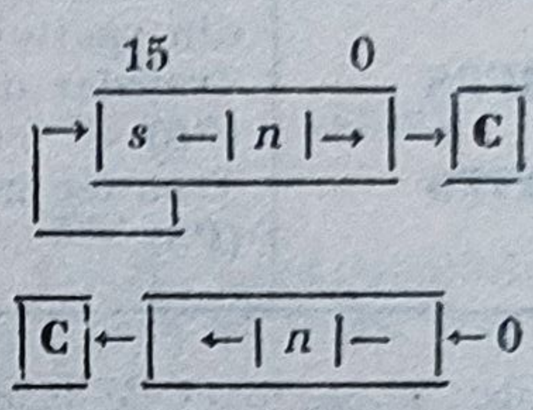
1	2	3	4	5
ASH	Deplasare aritmetică stînga/dreapta cu un număr de biţi	072RSS	* * * a	*srs* — ultimii 6 biţi conţin un număr *n*, care : $\|n\|$ = nr. poziţii deplasare semn *n* = semn deplasare (+ = la stînga) *dst* obligatoriu un registru 
ASHC	Deplasare aritmetică stînga/dreapta dublucuvînt, cu un număr de biţi	073RSS	* * * a	Similar cu **ASH**, deplasîndu-se conţinutul lui **R** şi **R** + 1, considerat un număr cu 32 biţi. **R** trebuie să fie par (dacă **R** este impar, deplasarea la dreapta devine rotaţie).

Tabela A.5

SET INSTRUCŢIUNI ÎN VIRGULA MOBILĂ SCURTĂ (FIS)

Mnemonica	Denumire instrucţiune	Cod operaţie (octal)	Poziţionare indicatori N Z V C	Funcţia
FADD	Adunare în virgulă mobilă (scurtă)	07500R	* * 0 0	**R** + 4, **R** + 6 ← [**R** + 4, **R** + 6] + + [**R**, **R** + 2] dacă rezultat $\geqslant 2^{-128}$; în caz contrar : **R** + 4, **R** + 6 ← 0
FSUB	Scădere în virgulă mobilă (scurtă)	07501R	* * 0 0	**R** + 4, **R** + 6 ← [**R** + 4, **R** + 6] − − [**R**, **R** + 2] dacă rezultat $\geqslant 2^{-128}$; în caz contrar : **R** + 4, **R** + 6 ← 0
FMUL	Înmulţire în virgulă mobilă (scurtă)	07502R	* * 0 0	**R** + 4, **R** + 6 × [**R** + 4, **R** + 6] − * [**R**, **R** + 2] dacă rezultat $\geqslant 2^{-128}$; în caz contrar : **R** + 4, **R** + 6 ← 0
FDIV	Împărţire în virgulă mobilă (scurtă)	07503R	* * 0 0	**R** + 4, **R** + 6 − [**R** + 4, **R** + 6]/[**R**, **R**+2] dacă rezultat $\geqslant 2^{-128}$; în caz contrar : **R** + 4, **R** + 6 ← 0

Tabela A.6

SET INSTRUCȚIUNI ÎN VIRGULĂ MOBILĂ LUNGĂ (FPP)

CONVENȚII DE NOTAȚII:

FPP = unitate de calcul în virgulă mobilă („Floating Point Processor")
FSS = element de adresă pentru operandul sursă, în modul de adresare FPP (similar cu SS din modul de adresare al UC)
FDD = idem, pentru operandul destinație (similar cu DD)
fsrs = conținut operand sursă al FPP
fdst = idem, operand destinație
FN = indicator de condiție negativă al FPP (similar cu N al UC)
FZ = idem, de condiție zero (similar cu Z)
FV = idem, de existență depășire (similar cu V)
FC = idem, de existență transport (similar cu C)
FD = indicator de simplă (FD = 0)/dublă (FD = 1) precizie a datelor reprezentate în virgulă mobilă
FL = indicator de simplă (FL = 0)/dublă (FL = 1) precizie de reprezentare a numerelor întregi la conversia între formatele virgulă fixă → virgulă mobilă (întreg simplă precizie = 1 cuvînt = 16 biți; întreg dublă precizie = 2 cuvinte = 32 biți)
VM = virgulă mobilă
AC = acumulator FPP
FPS = registrul de stare al FPP
FEC = codul de excepții al FPP (Floating Exception Code)
FEA = registrul de adresă pentru excepții FPP
M = mantisa
E = exponent

Mnemonic	Denumire instrucțiune	Cod operație (octal)	Poziționare indicatori FN FZ FV FC	Funcția
1	2	3	4	5
CFCC	Copiere valori indicatori de condiție	170000	— — — —	**N ← FN ; Z ← FZ V ← FV ; C ← FC**
SETF	Stabilire mod de lucru precizie simplă	170001	— — — —	**FD ← 0**
SETD	Stabilire mod de lucru precizie dublă	170011	— — — —	**FD ← 1**
SETI	Stabilire mod de lucru întregi simplă precizie	170002	— — — —	**FL ← 0**
SETL	Stabilire mod de lucru întregi dublă precizie	170012	— — — —	**FL ← 1**
LDFPS	Încărcare registru de stare FPP	1701FSS	— — — —	FPS ← *fsrs*
STFPS	Memorare conținut registru de stare FPP	1702FDD	— — — —	*fdst* ← FPP
STST	Memorare stare FPP (coduri de excepție și pointer de adresa de excepție)	1703FDD	— — — —	*fdst* ← FEC *fdst* + 2 ← FEA

Tabela A.6 (continuare)

1	2	3	4	5
CLRF **CLRD**	Şterge număr **VM** simplă/dublă precizie	1704FDD	0 1 0 0	*fdst* ← 0
TSTF **TSTD**	Testare număr **VM**, simplă/dublă precizie	1705FDD	* * 0 0	**FZ** ← 1 dacă E(*fdst*) = 0 **FZ** ← 0 în caz contrar **FN** ← 1 dacă fdst < 0 **FN** ← 0 în rest
ABSF **ABSD**	Formează valoarea absolută a numărului **VM**, simplă/dublă precizie	1706FDD	0 * 0 0	*fdst* ← -fdst. dacă *fdst* < 0 *fdst* ← 0 dacă E(*fdst*) = 0 *fdst* ← *fdst* în rest.
NEGF **NEGD**	Formează valoarea negativă a numărului **VM**, simplă/dublă precizie	1707FDD	* * 0 0	*fdst* ← — *fdst* dacă E(*fdst*) ≠ 0 *fdst* ← 0, în rest.
MULF **MULD**	Înmulţire numere **VM**, simplă/dublă precizie	171(AC)FSS	* * * 0	AC ← AC * *fsrs*
MODF **MODD**	Înmulţire numere **VM**, simplă/dublă precizie, cu transformare în întreg	171(AC+4)FSS	* * * 0	Se calculează produsul celor două numere **VM**, separă produsul în parte întreagă şi parte fracţionară, memorează una sau ambele părţi ca numere **VM**.
ADDF **ADDD**	Adunare numere **VM**, simplă/dublă precizie	172(AC)FSS	* * * 0	AC ← AC + *fsrs*
LDF **LDD**	Încărcare număr **VM**, simplă/dublă precizie	172(AC+4)FSS	* * 0 0	AC ← *fsrs*
SUBF **SUBD**	Scădere numere **VM**, simplă/dublă precizie	173(AC)FSS	* * * 0	AC ← AC — *fsrs*
CMPF **CMPD** **CMPD**	Compară număr **VM**, simplă/dublă precizie cu acumulatorul	173(AC+4)FSS	* * 0 0	**FN** ← 1 dacă *fsrs* < AC **FN** ← 0 în rest **FZ** ← 1 dacă *fsrs* = AC **FZ** ← 0 în rest
STF **STD**	Memorare număr **VM**, simplă/dublă precizie	174(AC)FDD	— — — —	*fdst* ← AC
DIVF **DIVD**	Împărţire număr **VM**, simplă/dublă precizie	174(AC+ 4)FSS	* * * 0	AC ← AC/*fsrs*
STEXP	Memorare exponent număr **VM**	175(AC)FDD	* * 0 0	*fdst* ← E(AC) — 200 (octal)
STCFI **STCFL** **STCDI** **STCDL**	Memorare şi conversie număr **VM**, simplă/dublă precizie în întreg simplă/dublă precizie	175(AC+4)FDD	* * * *	Se converteşte un număr **VM** din acumulator într-un număr întreg ce se memorează la *fdst*.

Tabela A.6 (continuare)

1	2	3	4	5
STCFD **STCDF**	Memorare și conversie număr **VM**, din simplă în dublă precizie și invers	176(AC)FDD	* * * 0	Conținutul acumulatorului se convertește din **VMFD** și se memorează în *fdst*, funcție de **FD** și **FL**.
LDEXP	Încărcare exponent	176(AC+4)DD	* * * 0	E(AC) ↔ *fsrs* + 200 (octal)
LDCIF **LDCID** **LDCLF** **LDCLD** **LDCDF**	Încărcare și conversie întreg precizie simplă/dublă în număr **VM**, precizie simplă/dublă	177(AC)DD	* * 0 0	Se convertește un întreg din *fsrs* într-un număr în **VM** ce se încarcă în acumulator, funcție de **FD** și **FL**
LDCFD	Încărcare și conversie număr **VM** din dublă în simplă precizie și invers	177(AC+4)FSS	* * * 0	Se convertește numărul **VM** din *fsrs* din dublă în simplă precizie și invers, cu care se încarcă acumulatorul

Tabela A.7

SUMAR AL EXTENSIILOR DE INSTRUCȚIUNI I-102F

Mnemonica	Instrucțiune	Cod operație
GPAT	Furnizare adresă început UCS și adresă configurație de date de inițializare	076700
DIS	Dezactivare cod rezervat, alocat anterior	076701
RES	Alocare cod rezervat unui microprogram dat	076702
GCOD	Furnizare informații privind un cod dat	076703
LDUC	Transfer de date din memoria principală în memoria de microprograme	076704
UNLD	Transfer de date din memoria de microprograme în memoria principală	076705
CKUC	Comparare conținut memorie de microprograme cu conținut memorie principală	076706

Tabela A.8

SETUL DE INSTRUCȚIUNI COMERCIALE (CIS)

NOTA ȘI CONVENȚII DE NOTAȚIE:

Operanzii acestor instrucțiuni sînt definiți prin *descriptori de șiruri* (de cifre zecimale sau de caractere) care pot fi localizați într-unul din următoarele două moduri:

- forma *registru* (mnemonicele instrucțiunilor nu se termină cu litera „I"), în care descriptorii se găsesc în registrele generale;
- forma *„in-linie"* (mnemonicele instrucțiunilor se termină cu litera „I"), în care descriptorii se găsesc în locațiile de memorie ce urmează imediat instrucțiunii.

srs — șir sursă; *dst* — șir destinație
NS — Descriptor și sursă
DD — Descriptor și destinație
DC — Descriptor set caractere
DP — Descriptor deplasare
Adr — adresa instrucțiunii CIS curente
A(e) — adresa (pointerul) elementului *e*
CZCM(P)S — cifra zecimală cea mai (puțin) semnificativă

Tabela A.8 (continuare)

Mnemonica	Denumire instrucțiune	Cod instr. (octal)	Operanzi	Poziționare indicatori N Z V C	Funcția
1	2	3	4	5	6
a) *Prelucrări șiruri cifre zecimale*					
ADDN	Adunare zecimale	076050	R0, R1=DS1	* * * 0	*dst* ← *srs 1*+*srs 2*
ADDP	Adunare zecimale impachetate	076070	R2, R3=DS2 R4, R5=DD		**R0÷R3** ← 0
ADDNI	Adunare zecimale	076150	Adr+2= = A(DS1)	* * * 0	*dst* ← *srs 1*+*srs 2*
ADDPI	Adunare zecimale	076170	Adr+4=A(DS2) Adr+6=A(DD)		
SUBN	Scădere zecimale	076051	R0, R1=DS1	* * * 0	*dst* ← *srs 2* — *srs 1*
SUBP	Scădere zecimale impachetate	076071	R2, R3=DS2 R4, R5=DD		**R0÷R3** ← 0
SUBNI	Scădere zecimale	076151	Adr+2=S(DS1)	* * * 0	*dst* → *srs 2* — *srs 1*
SUBPI	Scădere zecimale impachetate	076171	Adr+4=A(DS2) Adr+6=A(DD)		
CMPN	Comparare zecimale	076052	R0, R1=DS1	* * 0 0	**N** = 1 dacă *srs 1* < *srs 2*
CMPP	Comparare zecimale impachetate	076072	R2, R3=DS2		**Z** = 1 dacă *srs 1* = *srs 2* **R0÷R3** ← 0
CMPNI	Comparare zecimale	076152	Adr+2=A(DS1)	* * 0 0	**N** = 1 dacă *srs 1* < *srs 2*
CMPPI	Comparare zecimale impachetate	076172	Adr+4=A(DS2)		**Z** = 1 dacă *srs 1* = *srs 2*
ASHN	Deplasare aritmetică zecimale	076056	R0, R1=DS	* * * 0	*dst* ←
ASHP	Deplasare aritmetică zecimale impachetate	076076	R2, R3=DD R4=descriptor deplasare (DP): 0÷7: contor 8÷11: rotunjire 12÷15: 0		*srs* × 10 *contor* **R0, R1, R3** ← 0 Dacă *contor* > 0, deplasare spre CZCMS; dacă *contor* < 0, deplasare spre CZCMPS
ASHNI	Deplasare aritmetică zecimale	076156	Adr+2=A(DS1)	* * * 0	*dst* ←
ASHPI	Deplasare aritmetică zecimale impachetate	076176	Adr+4=A(DD) Adr+6=DP (vezi mai sus)		*srs* × 10 *contor* Dacă *contor* > 0, deplasare spre CZCMS; dacă *contor* < 0, deplasare spre CZCMPS.
MULP	Înmulțire zecimale impachetate	076074	R0, R1=DS1 R2, R3=DS2 R4, R5=DD	* * * 0	*dst* ← *srs 1* × *srs 2* **R0, R1, R3** ← 0
DIVP	Împărțire zecimale impachetate	076075	Idem	* * * 0	*dst* ← *srs 2*/*srs 1* **R0, R1, R3** ← 0

Tabela A.8 (continuare)

1	2	3	4	5	6
MULPI	Înmulțire zecimale împachetate	076174	Adr+2=A(DS1) Adr+4=A(DS2) Adr+6=A(DD)	* * * 0	*dst* ← *srs 1* × *srs 2*
DIVPI	Împărțire zecimale împachetate	076175	idem	* * * *	*dst* ← *srs 2*/*srs 1*

b) *Conversii șiruri zecimale*

CVTLN	Conversie întreg lung în zecimal	076057	R0, R1=DD R2, R3=întreg lung sursă	* * * 0	*dst* (șir zecimal) ← *srs* (întreg lung). **R2, R3** ← 0
CVTLP	Conversie întreg lung în zecimal împachetat				
CVTLNI	Conversie întreg lung în zecimal	076157	Adr+2=A(DD) Adr+4=A(întreg lung)	* * * 0	*dst* (șir zecimal) ← *srs* (întreg lung).
CVTLPI	Conversie întreg lung în zecimal împachetat	076177			
CVTNL	Conversie zecimal în întreg lung	076053	R0, R1=DS R2, R3=întreg lung destinație	* * * *	*dst* (întreg lung) ← *srs* (șir zecimal). **R0, R1** ← 0
CVTPL	Conversie zecimal împachetat în întreg lung	076073			
CVTNLI	Conversie zecimal în întreg lung	076153	Adr+2=A(DS) Adr+4=(întreg lung destinație)	* * * *	*dst* (întreg lung) ← *srs* (șir zecimal)
CVTPLI	Conversie zecimal împachetat în întreg lung	076173			
CVTNP	Conversie zecimal în zecimal împachetat	076055	R0, R1=DS R2, R3=DD	* * * 0	*dst* (zecimal împachetat) ← *srs* (zecimal). **R0, R1** ← 0
CVTPN	Conversie zecimal împachetat în zecimal	076054	idem	* * * 0	*dst* (zecimal) ← *srs* (zecimal împachetat).
CVTNPI	Conversie zecimal în zecimal împachetat	076155	Adr+2=A(DS) Adr+4=A(DD)	* * * 0	*dst* (zecimal impac.) → *srs* (zecimal). **R0, R1** ← 0
CVTPNI	Conversie zecimal împachetat în zecimal	076154	idem	* * * 0	*dst* (zecimal) ← *srs* (zecimal împachetat).

c) *Transfer șiruri de caractere*

MOVC	Transfer șir caractere	076030	R0, R1=DS R2, R3=DD	* * * *	*dst* ← *srs* R0 ← max. (0, lung. *srs* − lung. *dst*). **R1÷R3**←0
MOVRC	Transfer șir caractere, aliniat invers (dreapta)	076031	R4(0÷7) = caracter umplere		

Tabela A.8 (continuare)

1	2	3	4	5	6
MOVCI	Transfer șir caractere	076130	Adr+2=A(DS) Adr + 4=A(DD)	* * * *	*dst* ← *srs*
MOVRCI	Transfer șir caractere aliniat invers (dreapta)	076131	Adr+6(0÷7)= caracter umplere		
MOVTC	Transfer caracter translatat	076032	R0, R1=DS R2, R3=DN R4 (0÷7)= caracter umplere R5=A (tabel cu 256 caractere de translatare)	* * * *	*dst* ← *srs* translatat (fiecare caracter-sursă indexează tabelul de 256 caractere de translatare). **R0**← max. (0, lung. *srs* — lung. *dst*). **R1 ÷ R3** ← 0
MOVTCI	Transfer caracter translatat	076132	Adr+2=A(DS) Adr+4=A(DD) Adr+6(0÷7)= caracter umplere Adr+8=A(tabel cu 256 caractere de translatare)	* * * *	*dst* ← *srs* translatat (fiecare caracter-sursă indexează tabelul cu 256 caractere de translatare)
d) *Căutare în șiruri de caractere*					
LOCC	Localizare caracter	076040	R0, R1 = DS R4 (0÷7)= caracterul căutat	* * 0 0	Se caută caracterul indicat în șirul sursă În caz de succes: **BNE** = 1 În caz de insucțes: **BEQ** = 1
LOCCI	Localizare caracter	076140	Adr+2=A(DS) Adr+4(0÷7)= caracterul căutat R0, R1 = DS returnat		Idem, ca la **LOCC**
SKPC	Căutare caracter diferit	076041	R0, R1=DS R4(0÷7)= caracterul indicat	* * 0 0	Se caută în șirul-sursă primul caracter diferit de cel indicat
SKPCI	Căutare caracter diferit	076141	Adr+2=A(DS) Adr+4(0÷7) = caracter indicat R0, R1 = descriptor sub-șir rezultat	* * 0 0	Idem, ca la **SKPC**
SCANC	Căutare caracter dintr-un set indicat	076042	R0, R1=DS R4, R5=DC	* * 0 0	Se caută în șirul-sursă un caracter ce aparține setului indicat de caractere
SCANCI	Căutare caracter dintr-un set indicat	076142	Adr+2=A(DS) Adr+4=A(DC)	* * 0 0	Idem, ca la **SCANC**

Tabela A.8 (continuare)

1	2	3	4	5	6
SPANC	Căutare caracter diferit de setul indicat	076043	{R0, R1=DS R4, R5=DC	* * 0 0	Căutare în șirul-sursă caracterul diferit de setul indicat de caractere
SPANCI	Căutare caracter diferit de setul indicat	076143	{Adr+2=A(DS) Adr+4=A(DC) R0, R1=descriptor sub-șir rezultat	* * 0 0	Idem, ca la **SPANC**
CMPC	Comparare caractere	076044	{R0, R1=DS1 R2, R3=NS2 R4(0÷7)=caracter umplere	* * * *	Se compară șirurile *srs1* și *srs*, caracter cu caracter, pînă la prima diferență Test cu succes: **BEQ = 1** Test fără succes: **BNE = 1**
CMPCI	Comparare caractere	076144	{Adr+2=A(DS1) Adr+4=A(DS2)	* * * *	Idem, ca la **CMPC**
MATC	Identificare grup caractere	076045	{R0, R1=DS R2, R3=descriptor și obiect	* * 0 0	Se caută în șirul sursă coincidența cu întregul șir obiect
MATCI	Identificare grup caractere	076145	{Ad +2=A(DS) Adr+4=A(descriptor șir obiect) R0, R1 = descriptor sub-șir rezultat	* * 0 0	Idem, ca la **MATC**
c) *Încărcare descriptor instrucțiuni CIS*					
L2DR	Încărcare două perechi de descriptori	07602r	{R =A(adresa NS1) R +2=A(adresa DS2)	— — — —	**R0, R1** ← descriptor 1 **R2, R3** ← descriptor 2 (se aplică adresarea a **(Rr)** + de 2 ori)
L3DR	Încărcare trei perechi de descriptori	07606r	{Rr=A(Adresa DS1) Rr+2=A (Adresa DS2) Rr+4=A (Adresa DS3)	— — — —	**R0, R1** ← descriptor 1 **R2, R3** ← descriptor 2 **R4, R5** ← descriptor 3 (se aplică adresarea a **Rr)**+ de 3 ori)

Tabela A.9

RESTRICȚII DE UTILIZARE A MODURILOR DE ADRESARE

Mod	Registru	Nume mod	Operator/operand	Unde se găsește
000	X	Regirtru	Instrucțiune	Spațiul **I**
001	X	Registru indirect	Instrucțiune Dată	Spațiul **I** Spațiul **D**

Tabela A.9 (continuare)

Mod	Registru	Nume mod	Operator/operand	Unde se găseşte
010	0÷6	Autoincrementare	Instrucţiune Dată	Spaţiul I Spaţiul D
	7	Adresare imediată	Instrucţiune Dată imediată	Spaţiul I Spaţiul I
011	0÷6	Autoincrementare cu adresare indirectă	Instrucţiune Indirectare Dată	Spaţiul I Spaţiul D Spaţiul D
	7	Adresare absolută	Instrucţiune Adresare absolută Dată	Spaţiul I Spaţiul I Spaţiul D
100	0÷6	Autodecrementare	Instrucţiune Dată	Spaţiul I Spaţiul D
	7	A NU SE UTILIZA ACEST MOD		
101	0÷6	Autodecrementare cu adresare indirectă	Instrucţiune Indirectare Dată	Spaţiul I Spaţiul D Spaţiul D
	7	A SE UTILIZA ACEST MOD		
110	X	Indexat	Instrucţiune Index Dată	Spaţiul I Spaţiul I Spaţiul D
111	X	Indexat cu adresare indirectă	Instrucţiune Index Indirectare Dată	Spaţiul I Spaţiul I Spaţiul D Spaţiul D

Unde X — reprezintă orice registru general (R0÷R7)

SETUL DE CARACTERE CARE STAU LA BAZA CONSTRUCȚIILOR LIMBAJULUI MACRO

— literele A—Z și a—z ;
— cifrele 0—9 ;
— caracterele „.“ și „$“ ;
— caractere speciale cu funcții prestabilite precizate în următorul tabel :

Caractere	Funcție
:	terminator etichetă locală
::	terminator etichetă globală
=	operator pentru asignarea directă la simboluri locale
==	operator pentru asignarea directă la simboluri globale
%	indicator pentru registre (%0 ; %1 ; ... ; %7)
TAB	terminator de cîmp
blank	terminator de cîmp
#	indicator pentru expresie imediată
@	indicator de adresare pe multe nivele
(și)	caractere pentru desemnarea faptului că registrele conțin adrese (exemplu : (R2)).
,	separator în cîmpul operand
;	indicator care precede cîmpul comentariu
< și >	caractere pentru desemnarea expresiilor sau argumentelor (au rolul parantezelor cunoscute din matematică ; exemplu : < A + B >).
+	operator de adunare, autoincrementare sau producere a valorii pozitive
−	operator de scădere, autodecrementare sau producere a valorii negative
*	operator pentru înmulțire
/	operator pentru împărțire
&	operator logic SI
!	operator logic SAU
'	indicator pentru definirea unui caracter ASCII (exemplu : 'B)
"	indicator pentru definirea a două caractere ASCII (exemplu : "AR)
^	operator universal unar (exemplu : ^D129)
\	indicator pentru argumente numerice în apeluri de macroinstrucțiuni

DOCUMENTAȚIA FAMILIEI DE SISTEME DE OPERARE MIX

Familia de sisteme de operare **MIX** este însoțită de un set complet de manuale ce descriu structura și modul de utilizare al componentelor sistemului.

Documentația, elaborată în limba engleză, se distribuie pe suport magnetic în format final, listabil pe echipamente de imprimare.

1. Categorii de utilizatori

Documentele din setul de livrare **MIX** se adresează, parțial sau total, unor anumite categorii de utilizatori, în funcție de calitatea acestora.

Sistemul **MIX** definește mai multe categorii de utilizatori:

- inginer de sistem (**IS**) = responsabil cu buna funcționare generală a sistemului. **IS** reconfigurează sistemul, testează buna lui funcționare și execută toate operațiile de actualizare ale sistemului;
- programator de sistem (**PS**) = responsabil cu întreținerea, modificarea și/sau dezvoltarea în continuare a sistemului;
- utilizator (**U**) = responsabil cu dezvoltarea programelor sale de aplicație;
- operator (**O**) = oricine utilizează sistemul **MIX** pentru a efectua o lucrare. Categoria **O** poate include și personalul însărcinat cu executarea unor operații de rutină, sub directa conducere a **IS**.

Granițele între categorii sînt foarte elastice și depind de fiecare instalație în parte.

2. Categorii de manuale

Manualele sistemului de operare **MIX** sînt împărțite pe categorii, în funcție de tipul informațiilor pe care le conțin..

Se definesc următoarele categorii:

- Manuale de referință și utilizare sistem
- Manuale privind limbajele de programare
- Manuale privind administrarea datelor
- Manuale privind produsele de comunicații.

a) **Manuale de referință și utilizare sistem MIX/MIX-PLUS**

1. Catalog de documentație
2. Îndrumar pentru începători
3. Concepte și funcții ale familiei de sisteme de operare **MIX**
4. Monitor
5. Interfața operator-sistem **MCL**
6. Interfața operator-sistem **DCL**
7. Operarea sistemului în regim multi-utilizator
8. Fișiere indirecte de comenzi
9. Operarea minicalculatoarelor românești
10. **FCS** — Manual de programare
11. **RMS** — Manual de programare
12. **RMS** — Manual de utilizare
13. Drivere de intrare/ieșire
14. Drivere de timp real
15. Îndrumar de scriere a unui driver de intrare/ieșire
16. Subsisteme de depanare
17. Biblioteci sistem
18. Subsistem de înregistrare a erorilor

19. Subsistem de testare integrată
20. Subsistem de diagnosticare
21. Analiza căderilor sistem
22. Subsistem de gestiune cozi
23. Îndrumar de scriere a unui task ajutător de control
24. Subsisteme de contabilizare
25. Utilitare pentru dezvoltarea de programe
26. Utilitare pentru gestiunea volumelor
27. Utilitare pentru manipularea fişierelor
28. Utilitare **RMS**
29. Manualul inginerului de sistem
30. Proceduri de verificare a sistemului de operare
31. Editor de documentaţie
32. Editor de texte mod ecran
33. Macroasamblor
34. Editor de legături

MIX-RT/MIX/MIX-PLUS

1. Note de versiune, specifice fiecărui sistem

MIX-PLUS

1. Proceduri de reconfigurare
2. Subsistem batch

b) **Manuale privind limbajele de programare**

FORTRAN IV

1. Manual de programare
2. Manual de utilizare
3. Biblioteca matematică **FORTRAN**

FORTRAN 77

1. Manual de programare
2. Manual de utilizare

BASIC-PLUS

1. Manual de utilizare

BASIC-PLUS-2

1. Manual de programare
2. Manual de utilizare
3. Biblioteca matematică **BASIC-PLUS-2**

COBOL

1. Manual de programare
2. Manual de utilizare

COBOL-81

1. Manual de programare
2. Manual de utilizare

ADA

1. Manual de programare
2. Manual de utilizare

C

1. Manual de programare
2. Manual de utilizare

RTL/2

1. Manual de programare
2. Manual de utilizare

PROLOG

1. Manual de programare
2. Manual de utilizare

PASCAL

1. Manual de programare
2. Manual de utilizare
3. Utilitare **PASCAL**

c) **Manuale privind administrarea datelor**

1. Manual de utilizare **SORT**
2. Manual de utilizare **FMS**
3. Manual de utilizare **DATATRIEVE**

d) **Manuale privind produsele de comunicaţii**

MININET/MIX

1. Concepte şi facilităţi ale reţelei **MININET/MIX**
2. Comunicarea între task-uri
3. Programe de control reţea
4. Sistemul de gestiune a transmisiilor
5. Comunicarea între terminale
6. Încărcarea task-urilor/sistemelor la/de la distanţă
7. Controlul task-urilor la distauţă
8. Accesul fişierelor la distanţă
9. Utilitare de reţea
10. Protocolul funcţiilor de întreţinere comunicaţii
11. Protocolul funcţiilor de acces la date
12. Drivere de reţea
13. Protocoale de comunicaţii

INTERNETS

1. Emulator IBM 2780/3780
2. Emulator IBM 3271

Opţiuni de comunicaţie

1. Operarea şi programarea **KMC**
2. Manual de programare **COMM IOP-DUP**
3. Manual de programare **COMM IOP-DZ**
4. Manual de programare **MIX/K**

Lucrări apărute la Editura Tehnică în 1991 :

L. Dumitrașcu, T. Sperlea, C. Marinoiu
dBASE II, III, III Plus, IV – în conversații și 4 sinteze

R. W. Hockney, C. R. Jesshope
Calculatoare paralele
Arhitectură, algoritmi, programare *traducere din lb. engleză*

A. Petrescu ș.a.
abc de calculatoare personale

În curs de apariție

V. Petrovici
PARADOX
Ghid de utilizare... și nu doar atît

Gh. Samoilă
PC 386 – o nouă generație de calculatoare personale

B. Vlada ș.a.
Grafica pe calculator în limbajele Pascal și C

M. Zaharia
Învățăm să programăm

D. Somnea, T. Vlăduț
Calculatoare personale – programare în ASSEMBLER

V. Petrovici, F. Goicea
Calculatoare personale
Programarea în limbajul C

T. Bălănescu ș.a.
Programarea în Pascal și Turbo Pascal

Fl. Păunescu, D. Golești eanu
Sisteme cu prelucrare distribuită și aplicațiile lor

Seria Aide-mémorie :
- Memento Turbo Pascal
- Memento UNIX

Lucrarea își propune să ofere unui cerc larg de cititori (utilizatori, programatori, studenți) informațiile generale și specifice care să le permită atît o utilizare eficientă a minicalculatoarelor românești din familiile INDEPENDENT și CORAL, cît și elaborarea de programe aplicative care să exploateze pe cît mai bine multiplele facilități ale acestor echipamente de calcul.